



Today’s Talk will be hosted in **Chinese**

For English audience

- * SGL office hour: <https://www.youtube.com/watch?v=NS9N4ZHtAJU>
- * Friday 10am PST, English version hosted by agent lab

Advancing Token Productivity with Agent Loops

How to effectively transform tokens into productivity.

Ligeng Zhu

Research Scientist, NVIDIA Research

2026-Jan

Name	\$ limit ▾	MTD usage
Ligeng Zhu	—	\$22,254.15

2026-Mar

Name	\$ limit ▾	MTD usage
Ligeng Zhu	—	\$9,100.51

2026-Feb

Name	\$ limit ▾	MTD usage
Ligeng Zhu	—	\$6,014.33

2026-Apr

Your spend limit		
Your spend limit for this workspace		
Name	\$ limit ▾	MTD usage
Ligeng Zhu	-	\$40,671.06



About Me

Multi-Modal VLMs / Scalable and Efficient Agents



Ligeng Zhu

Researcher, NVIDIA Research

Supervised by Prof. Song Han

Research focus

- Agents for L2~L4 level Automatic Research & Engineering
- Scalable and Efficient Agent Harness and Swarms
- Efficient Multi-Modal VLMs

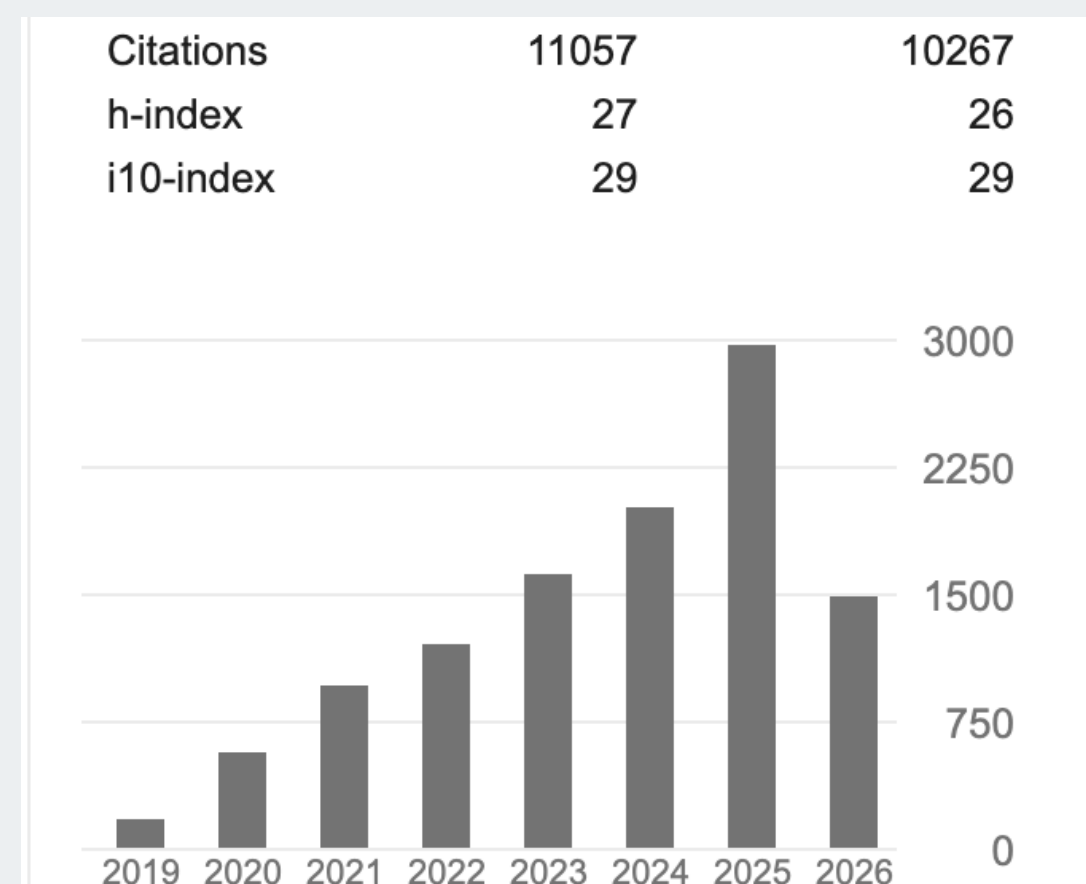
Core Author Projects at NVR

1. KDA: Kernel Develop Agents for BlackWell
2. JetAutoResearch: A token-efficient harness for agent-driven research
3. Parason: Revealing Subtask- and Trial Parallelism in LLM Reasoning
4. OckBench: Measuring the Efficiency of LLM Reasoning
5. EGM: Efficient Visual Grounding Language Models
6. Fast-dVLM: Efficient Block-Diffusion VLM via Direct Conversion from Autoregressive VLM
7. NVILA: Efficient Frontier Visual Language Models

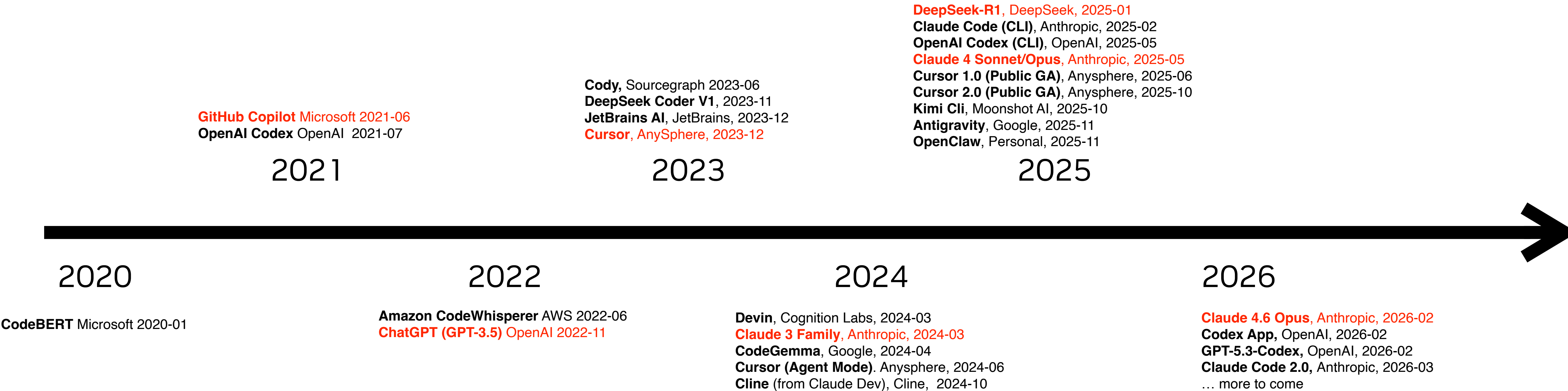
Core Author Projects at MIT

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

- Achieved 10k cites this year



The Progress of AI Coding



AI Coding Evolves So Quickly

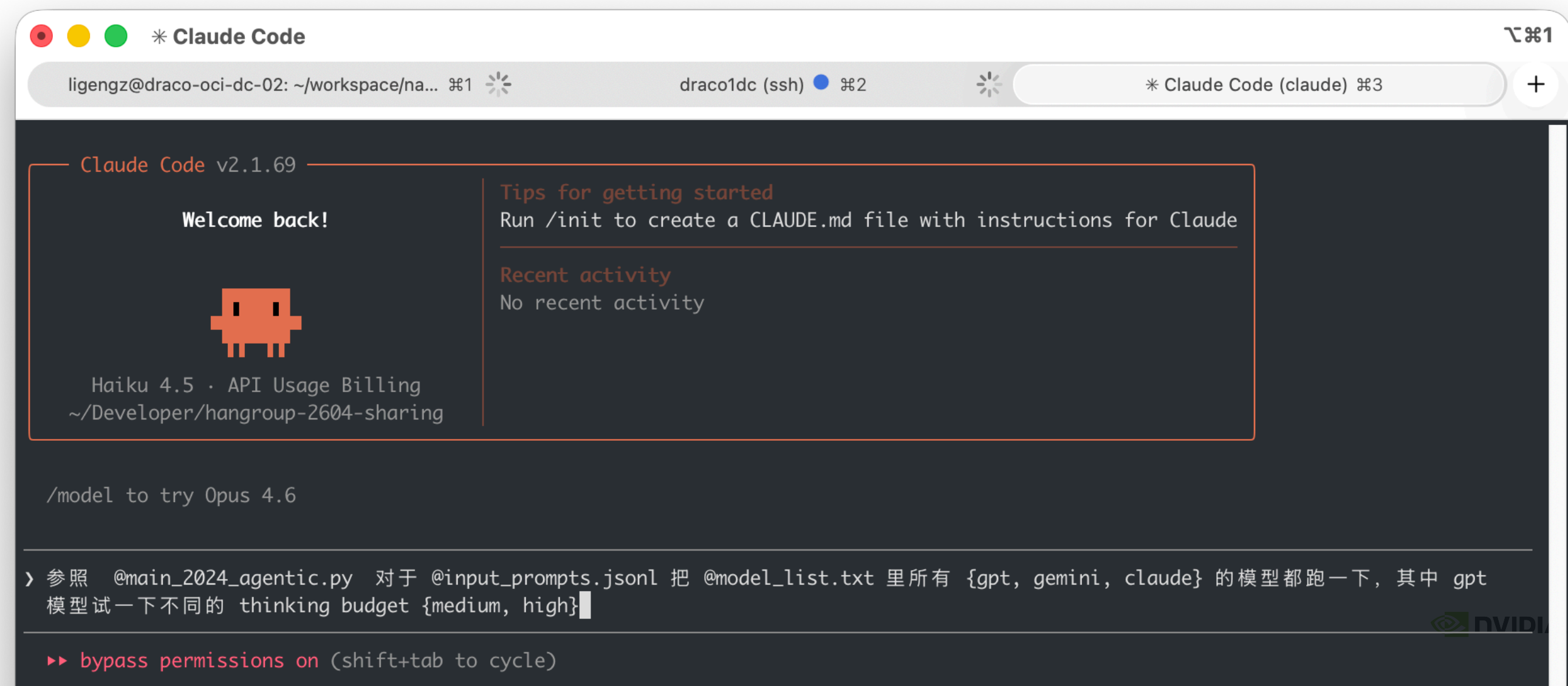
- The Power of Scaling
 - **More Knowledge** — the details/grammar/syntax/document of common software
 - **Longer Context** — memorize the whole conversation and not miss any clues for bugs
 - **Better Instruction Following** — knows what to do and what not to do
 - **More Accurate Function Calling** — from "assistants" (tools that suggest) to "agents" (tools that act)
 - **More Intelligent Reasoning** — decompose a complex task into multiple simple ones before execution
 -
- In April 2026, we are at the dawn of **Software AGI**



The Trends of AI Coding

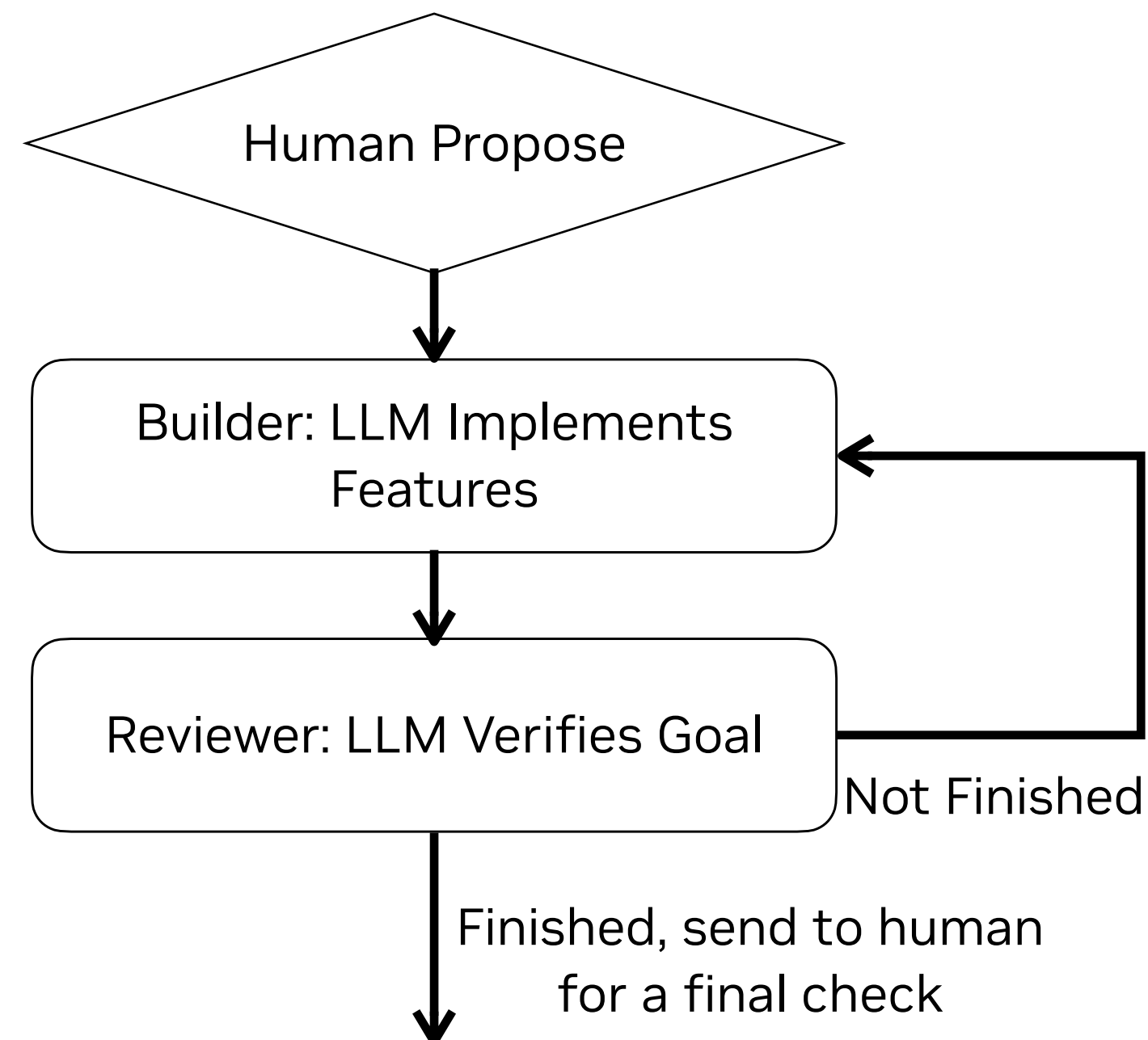
- 2020–2022 (**Autocomplete Era**): Focused on "Ghost Text"—finishing the line of code you were currently typing.
- 2023 (**Contextual Integration**): Focused on "RAG" (Retrieval-Augmented Generation)—understanding your whole folder, not just one file.
- 2024 (**The Agentic Shift**): from "assistants" (tools that suggest) to "agents" (tools that act). Focused on "Action"—running tests, fixing bugs autonomously in the terminal, and managing deployments.
- 2025 (**Reasoning, Multi-Step Agents**): The release of DeepSeek-R1 in January 2025 proved that reinforcement learning could enable open-weight models to compete with proprietary systems in complex multi-step logic.
- Mid-2025 to Early-2026 (**Autonomous Operations**): Tools that handle the entire engineering workflow from a ticket to merged pull request. Multiple parallel agents cooperate as a team/group/company, assign/delegate/verify the tasks asynchronously and in parallel.

```
main_2022.py > ...
1  from openai import OpenAI
2  import os
3  import json
4
5  api_key = os.environ.get("NVIDIA_API_KEY")
6
7  client = OpenAI(
8      api_key=api_key,
9      base_url="https://inference-api.nvidia.com"
10 )
11
12 with open("input_prompts.jsonl", "r") as f:
13     for line in f:
14         data = json.loads(line)
15         prompt = data["prompt"]
16         response = client.chat.completions.create(
17             model="llama3-8b-8192",
18             messages=[{"role": "user", "content": prompt}]
19         )
```



Agent Loops for Autonomous Operations

- Previous **vibe coding**: **single** agent session, **many** prompts
- Current **auto coding**: **many** agent sessions, **singe** prompt
- **Core**: Loop with Agents/LLMs!
 - Forget about GUI, move to CLI, chain agents inputs and outputs, then design your own workflow.

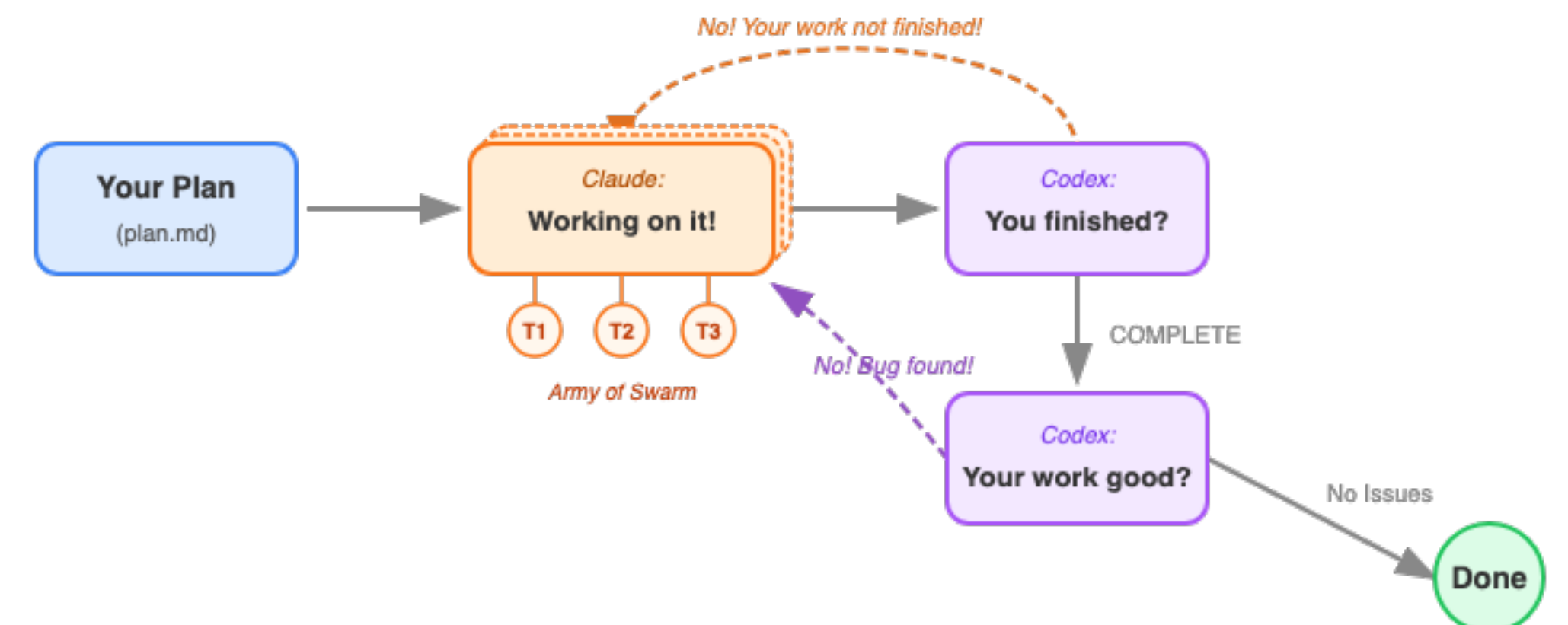


1. Flexibly change the Builder and Reviewer (Opus, GPT, Gemini, Codex, Kimi).
2. Freely change the workflow as you likes, for example
 1. Implement different parts w/ different LLMs (Gemini for frontend, GPT for backend, Claude for networking)
 2. Verification by multiple rounds / voting (accepted only when Gemini + GPT + Opus all agrees).

Such flexibility is not gonna achieved by GUIs.
Agent Flow **minimizes the human efforts**.

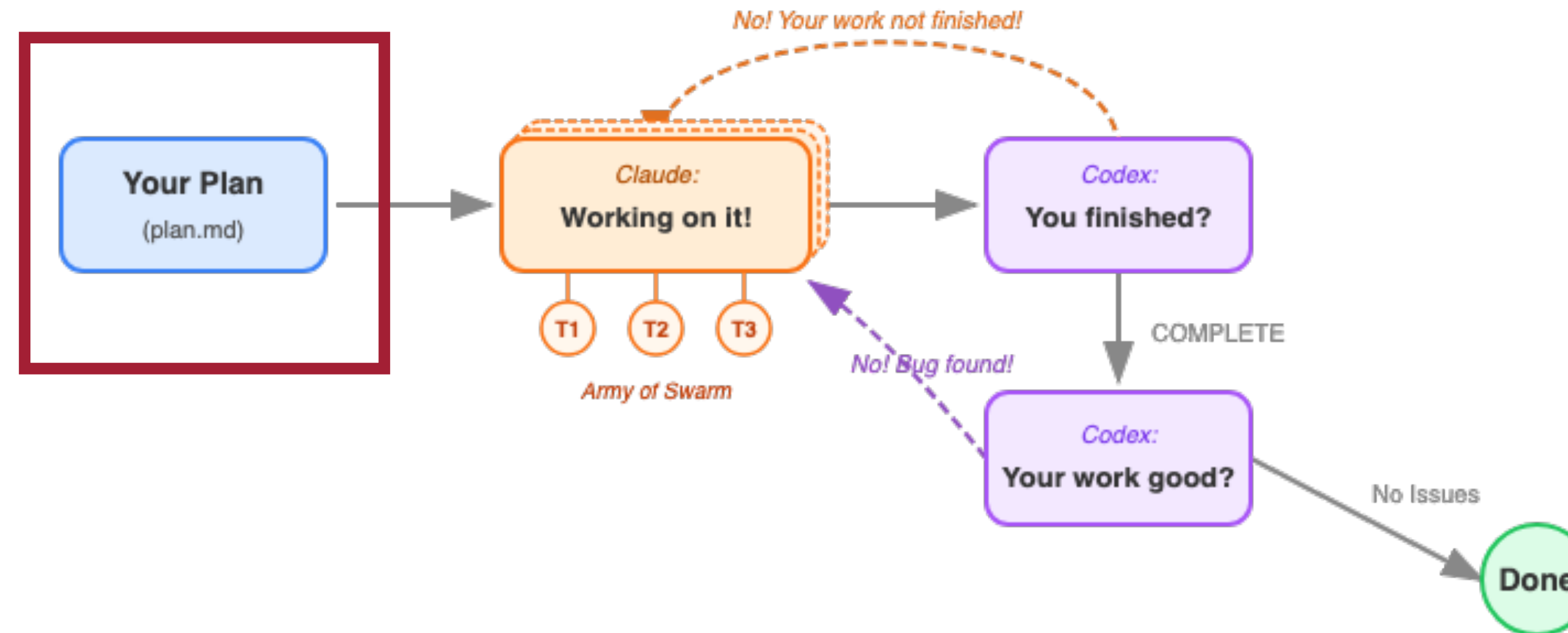
Advancing Modern Agent Loops

- Agent Loop is bit like real-world RL (input from human, mid reward from llm, and final judge by human)
 - Reinforcement Learning with Code Review
- <https://github.com/humana-org/humanize>
 - Side project from Sihao Liu (UCLA PhD -> NVR)
- Core Loops:
 - **Iteration over Perfection** -- Instead of expecting perfect output in one shot, Humanize leverages continuous feedback loops where issues are caught early and refined incrementally.
 - **One Build + One Review** -- Claude implements, Codex independently reviews. No blind spots.
 - **Begin with the End in Mind** -- Before the loop starts, Humanize verifies that you understand the plan you are about to execute. The human must remain the architect.
- Humanize is a pro-max-ultra-mythos version of AutoResearch.



Advancing Modern Agent Loops

- git clone <https://github.com/fla-org/flash-linear-attention>
- vim draft.md
- > read <https://arxiv.org/pdf/2510.26692>, based the repo, implement Kimi Delta Attention in cuda



cat draft.md

- read <https://arxiv.org/pdf/2510.26692>, based the repo, implement Kimi Delta Attention in cuda

Advancing Modern Agent Loops

- `git clone https://github.com/fla-org/flash-linear-attention`
- `vim draft.md`
- `> read https://arxiv.org/pdf/2510.26692`, based the repo, implement Kimi Delta Attention in cuda
- **`> /humanize:gen-plan --input draft.md --output docs/plan.md`**

* Running... (4m 25s · ↓ 13.3k tokens · thought for 10s) | □ Phase 2: Relevance check via agent

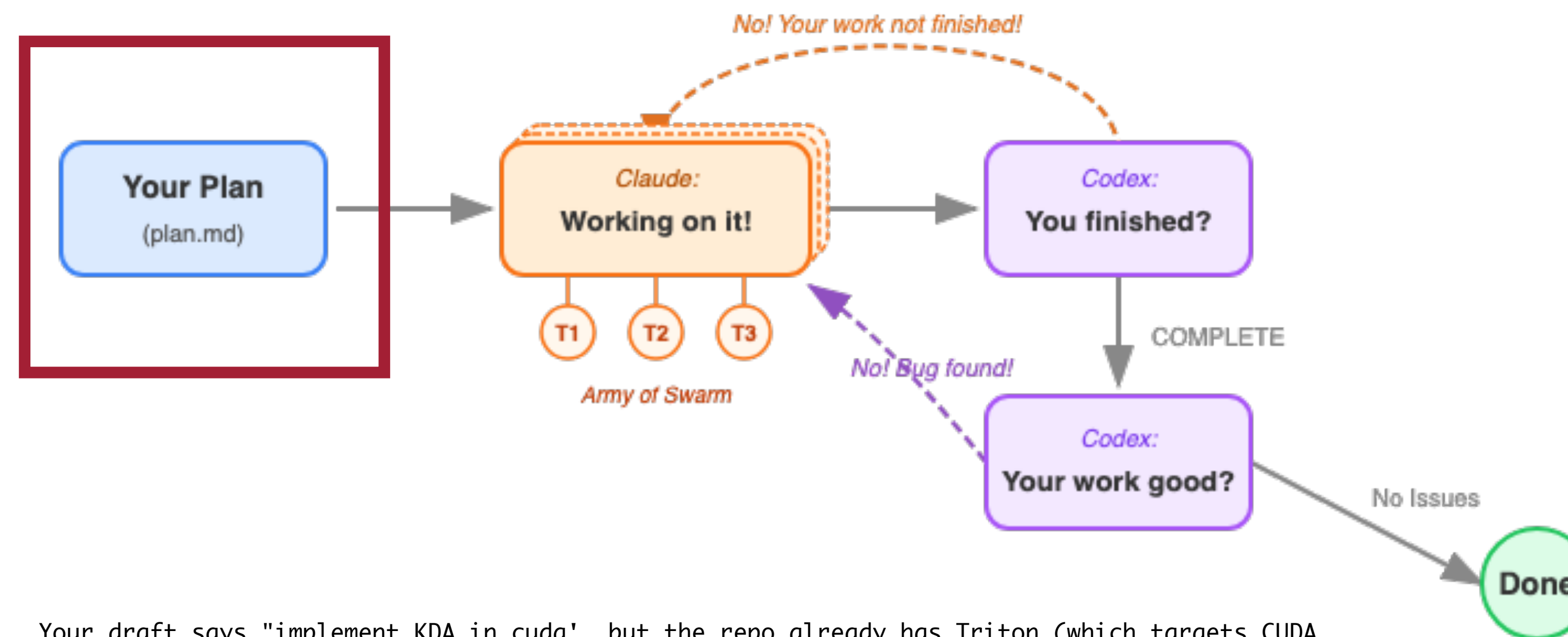
□ Phase 3: Codex first-pass analysis

□ Phase 4: Claude candidate plan v1

□ Phase 5: Iterative convergence loop with

second

□ Phase 6: Issue and disagreement resolution



Your draft says "implement KDA in cuda", but the repo already has Triton (which targets CUDA GPUS) and an external CUTLASS 'FlashKDA' backend that's inference-only with bf16 + K=V=128 + no GVA. What scope should the new CUDA backend cover?

> 1. Training (fwd + bwd) New in-tree CUDA C++ / CUTLASS backend with full backward pass (grads for q, k, v, g, beta, A_log, dt_bias). This is the differential value vs existing inference-only FlashKDA. Largest effort but unlocks training speedups.

> 2. Inference only. Forward-only CUDA backend; broader inference coverage than FlashKDA. Smaller effort but overlaps with existing FlashKDA on its shapes; differential is breadth (dtypes, head dims, GVA, etc.).

> 3. Replace Triton entirely Remove Triton implementation and use only CUDA. Highest risk; likely loses GVA/varlen/CP coverage. NOT recommended.

4. Type something.

Advancing Modern Agent Loops

- `git clone https://github.com/fla-org/flash-linear-attention`
- `vim draft.md`
- `> read https://arxiv.org/pdf/2510.26692`, based the repo, implement Kimi Delta Attention in cuda
- `> /humanize:gen-plan --input draft.md --output docs/plan.md`
- **`vim draft.md # add <cmt> to comment`**
- **`/humanize:refine-plan --input docs/plan.md`**

The implementation is additive: the existing Triton kernels in ``fla/ops/kda/`` and the existing external ``FlashKDABackend`` (inference-only) remain in place and continue to win backend selection on their respective supported shapes.

<cmt> Support NVFP4 on Blackwell as well </cmt>

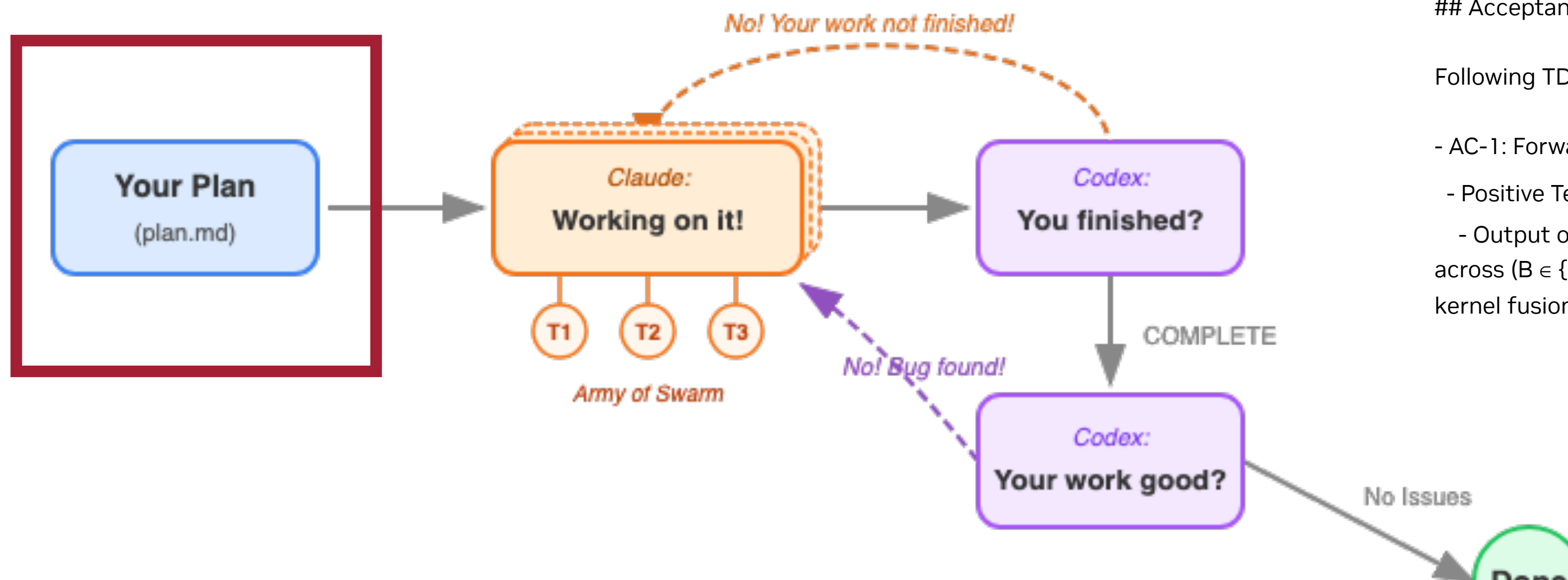
v1 Support Matrix (explicit)

- dtype: bf16
- head dims: $K = V = 128$
- path: chunk only (no ``fused_recurrent`` CUDA path in v1)
- chunk_size $\in \{32, 64\}$
- safe_gate = True with lower_bound in $[-5, 0]$
- state_v_first = True
- features supported in-kernel: L2-norm fusion, beta-sigmoid fusion, per-key-dim gate fusion, ``initial_state`` and ``output_final_state`` (both True and False)
- training: full backward gradients for q, k, v, g, beta, A_log, dt_bias
- features explicitly rejected at predicate (registry falls through to Triton): GVA ($HV \neq H$), varlen / ``cu_seqlens``, CP-active state, ``state_v_first=False``, ``allow_neg_eigval=True``, ``return_intermediate_states=True``, $K \neq 128$ or $V \neq 128$, ``safe_gate=False``, dtype $\neq$ bf16, ``initial_state.requires_grad=True``, ``torch.is_grad_enabled() == False``, no tensor with ``requires_grad=True``

Acceptance Criteria

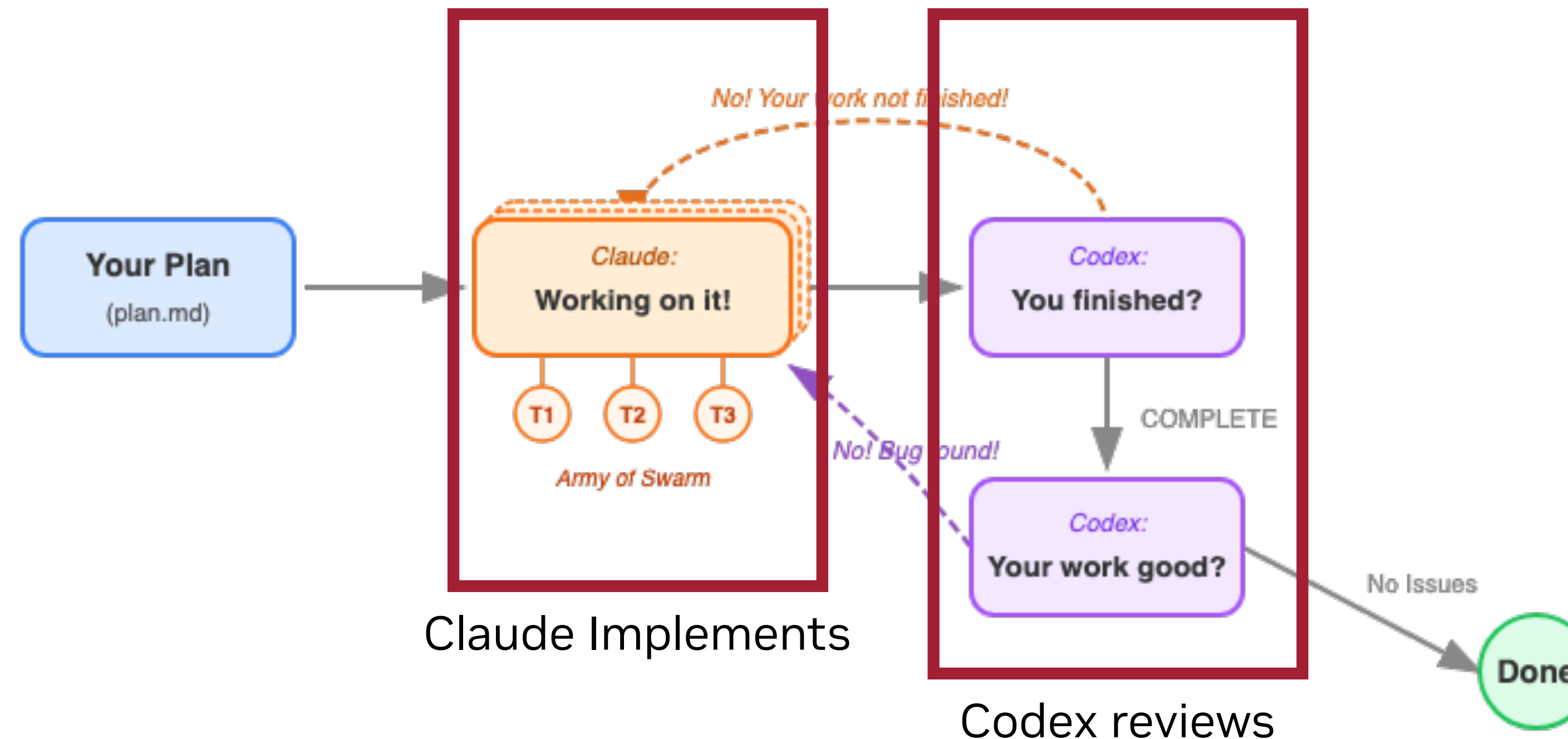
Following TDD philosophy, each criterion includes positive and negative tests for deterministic verification.

- AC-1: Forward numerical parity with reference implementations on the v1 matrix.
 - Positive Tests (expected to PASS): **<cmt> $\text{rtol} < 1e-4$ </cmt>**
 - Output of the new CUDA backend matches ``naive_chunk_kda`` within ``max-abs  $\leq 5e-3$ `` and ``rms  $\leq 1e-3$ `` across ($B \in \{1, 2, 4\}$, $T \in \{512, 2048, 8192\}$, $H \in \{4, 8\}$, $K = V = 128$, chunk_size $\in \{32, 64\}$, safe_gate=True, all in-kernel fusions ON, both with and without ``initial_state``).



Advancing Modern Agent Loops

- `git clone https://github.com/fla-org/flash-linear-attention`
- `vim draft.md`
- `> read https://arxiv.org/pdf/2510.26692`, based the repo, implement Kimi Delta Attention in cuda
- `> /humanize:gen-plan --input draft.md --output docs/plan.md`
- `vim draft.md # add <cmt> to comment`
- `/humanize:refine-plan --input docs/plan.md`
- **`/humanize:start-rlcr-loop docs/plan.md`**



Advancing Modern Agent Loops

Different Level of Vibe Coding

Vibe Code / Vibe Research =

Model (opus/gpt/deepseek/kimi) X

Tool (claudecode/codex/opencode) X

Flow (humanize/autoresearch/ralphloop)

Agent = Model + Tool

Harness = Tool + Flow

Model-level
(1~10mins)

```
response = client.responses.create(  
    model="gpt-5.5",  
    input="Profile FP8MoE and improve the  
    implementation."  
)  
# act blabla
```

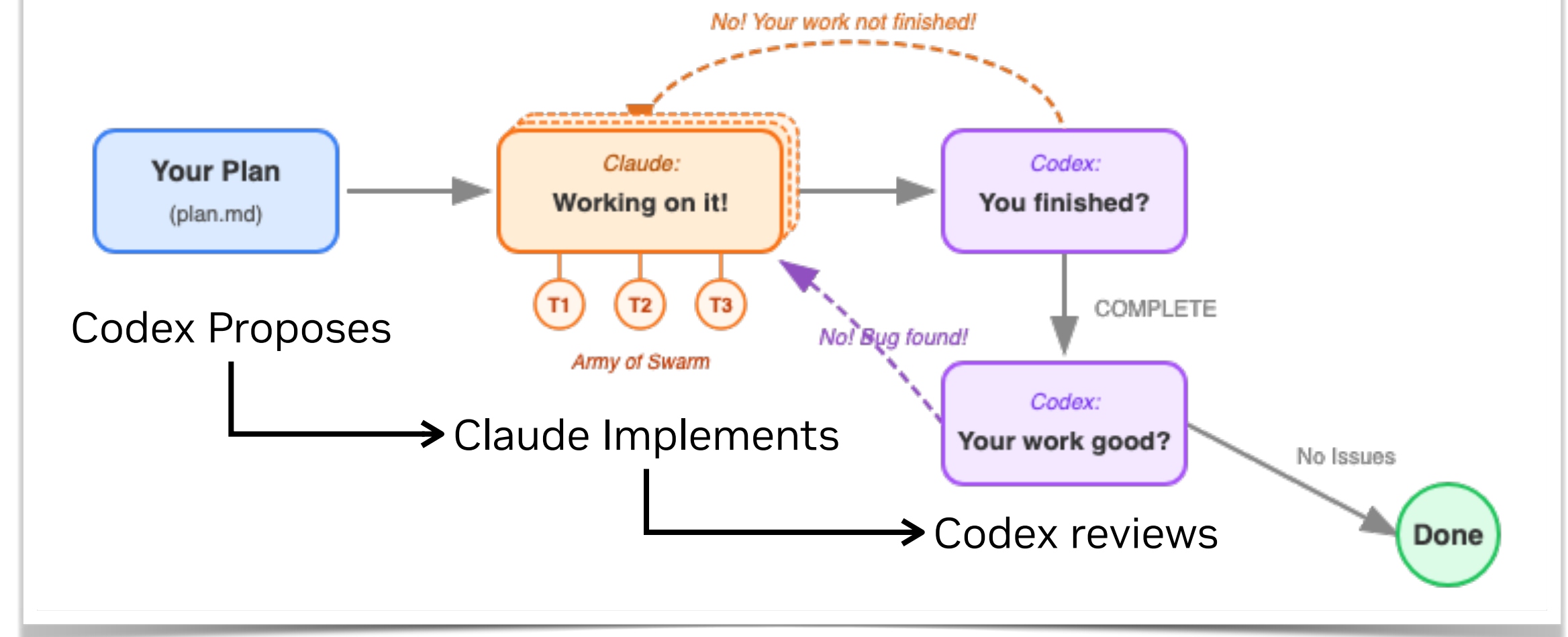
Tool-level
(10mins ~ hours)

```
>_ OpenAI Codex  
| permissions: YOLO mode  
  
> Profile FP8MoE @fp8_moe.py and improve  
the implementation
```

Flow-level
(days ~ weeks)

```
> /humanize:gen-plan --input draft.md --  
output docs/plan.md  
> /humanize:refine-plan --input docs/  
plan.md  
> /humanize:start-rlcr-loop docs/plan.md
```

Humanize 1.0





KDA: Kernel Develop Agents for BlackWell

Mastering the Top3 over >200 teams on
FlashInfer Kernel Contest at MLSys'26.

Ligeng Zhu

Research Scientist, NVIDIA Research

Method overview of KDA

Modern SOTA LLMs are powerful, but to unleash its full power, we need Model X Tool cooperation.

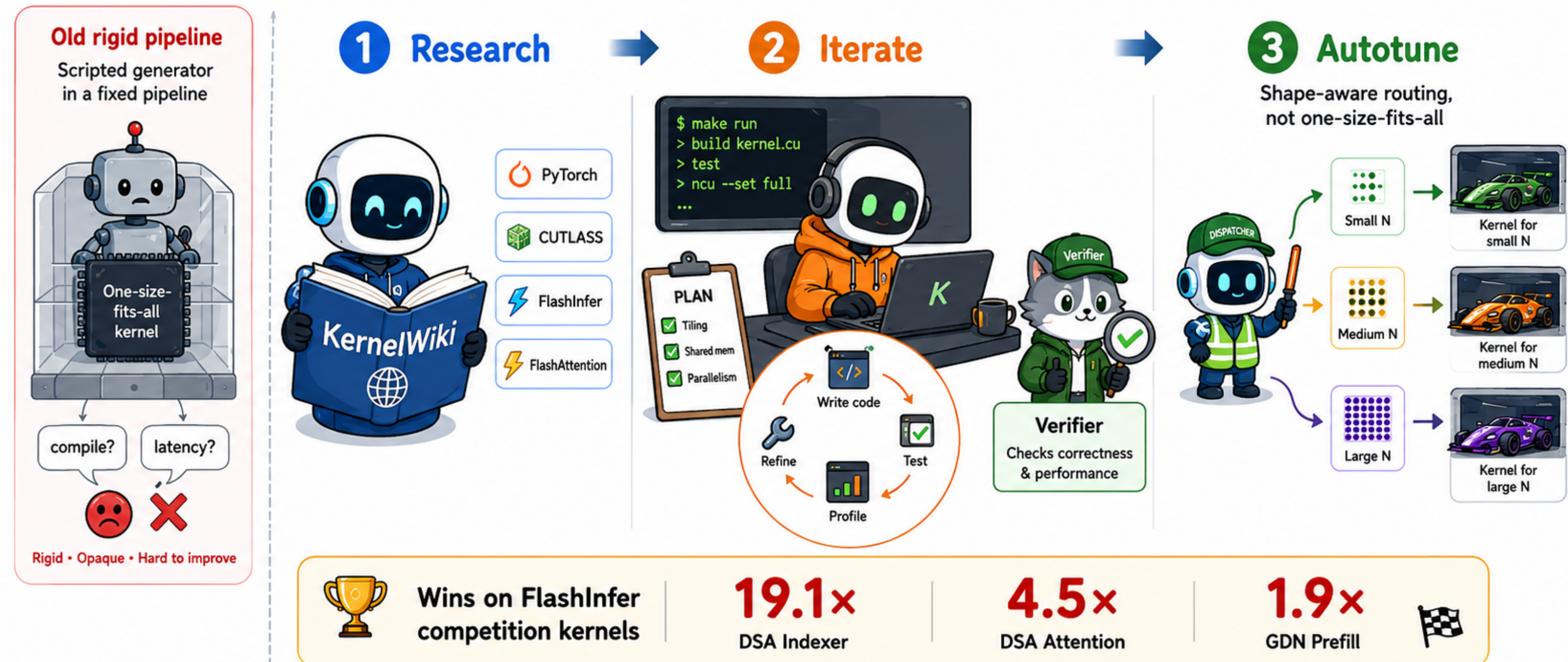
1 Humanize: Long running agents loops

2 KernelWiki: A curated implementation reference from SGLang, vLLM, TRTLLM

3 NCU-Profile.skill: analyze the detailed roofline and performance trace.

Developing GPU Kernels with Agentic Loops

Humanize Kernel Agent Loop lets an AI agent build fast GPU kernels like a human engineer.

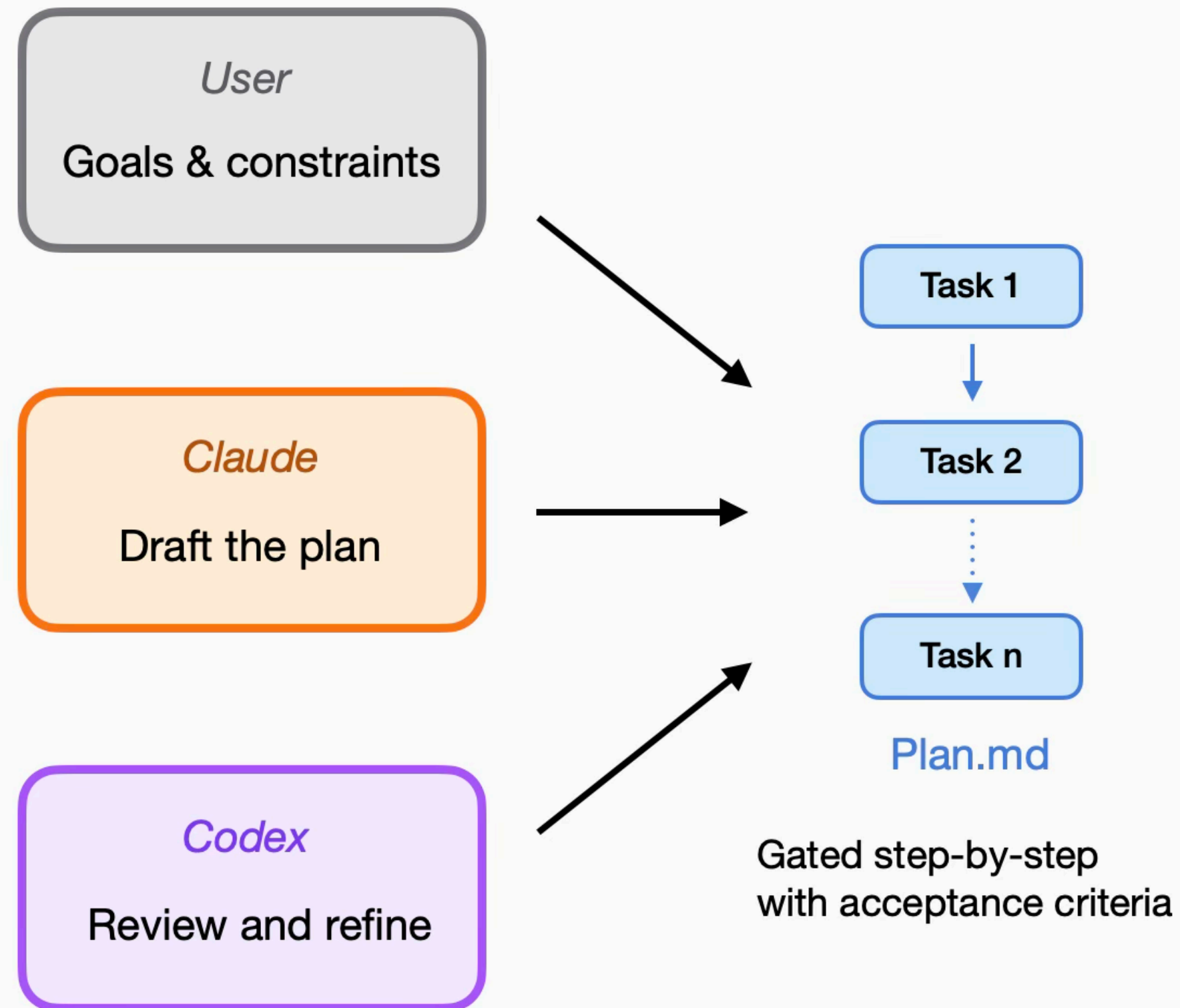


Knowledge + profiling + verification + autotuning = faster kernels.

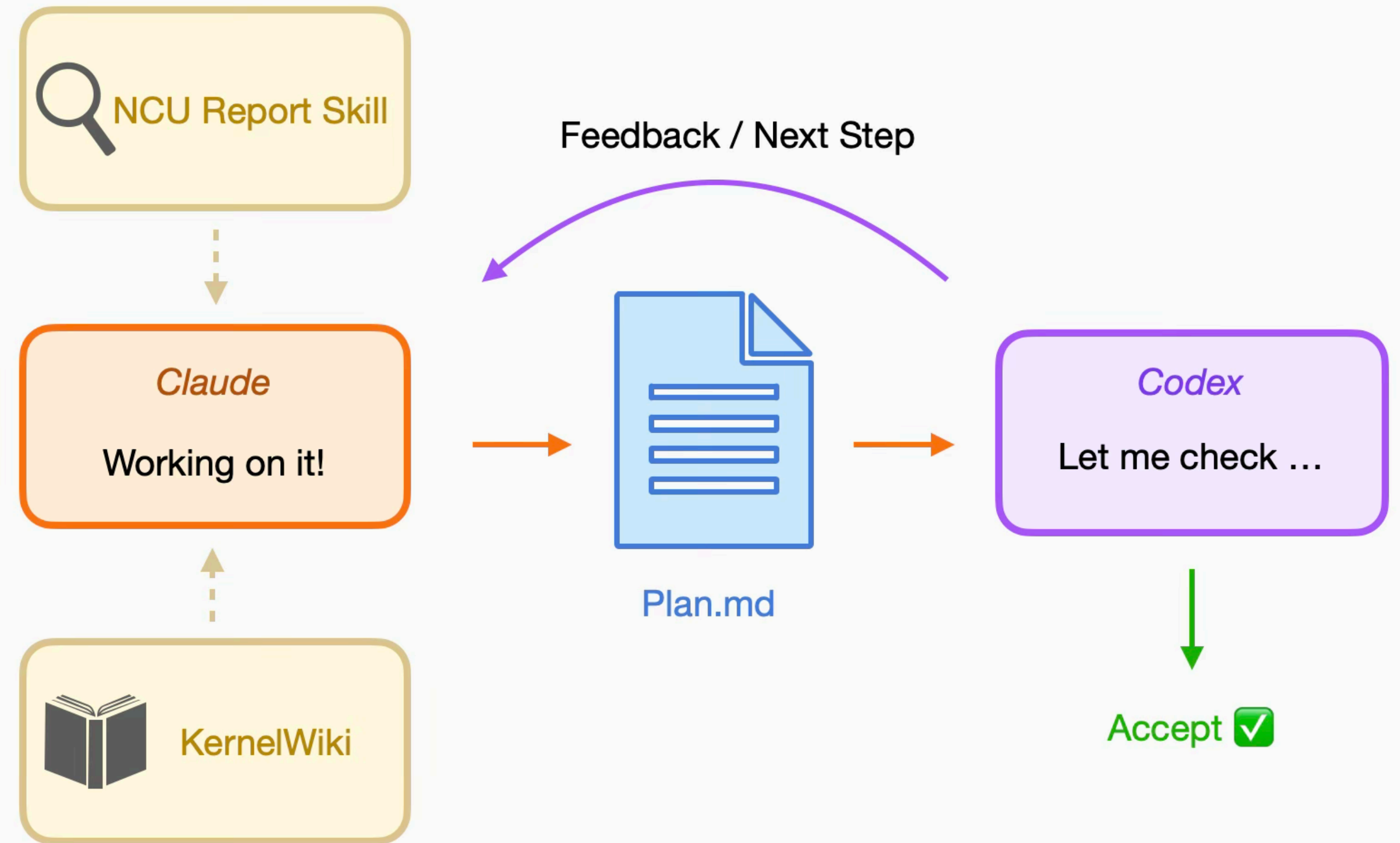
Humanize Agentic Loop

Coder and verifier work inside the real development environment

(A) Planning Phase



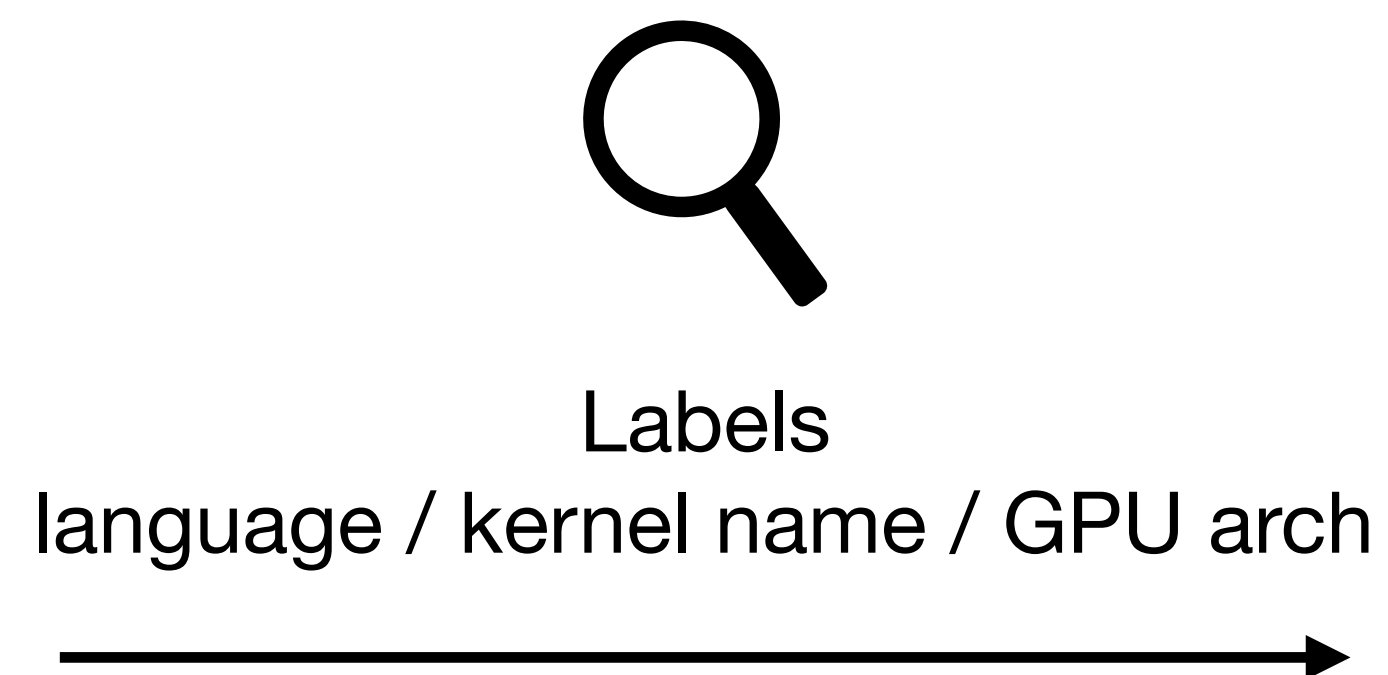
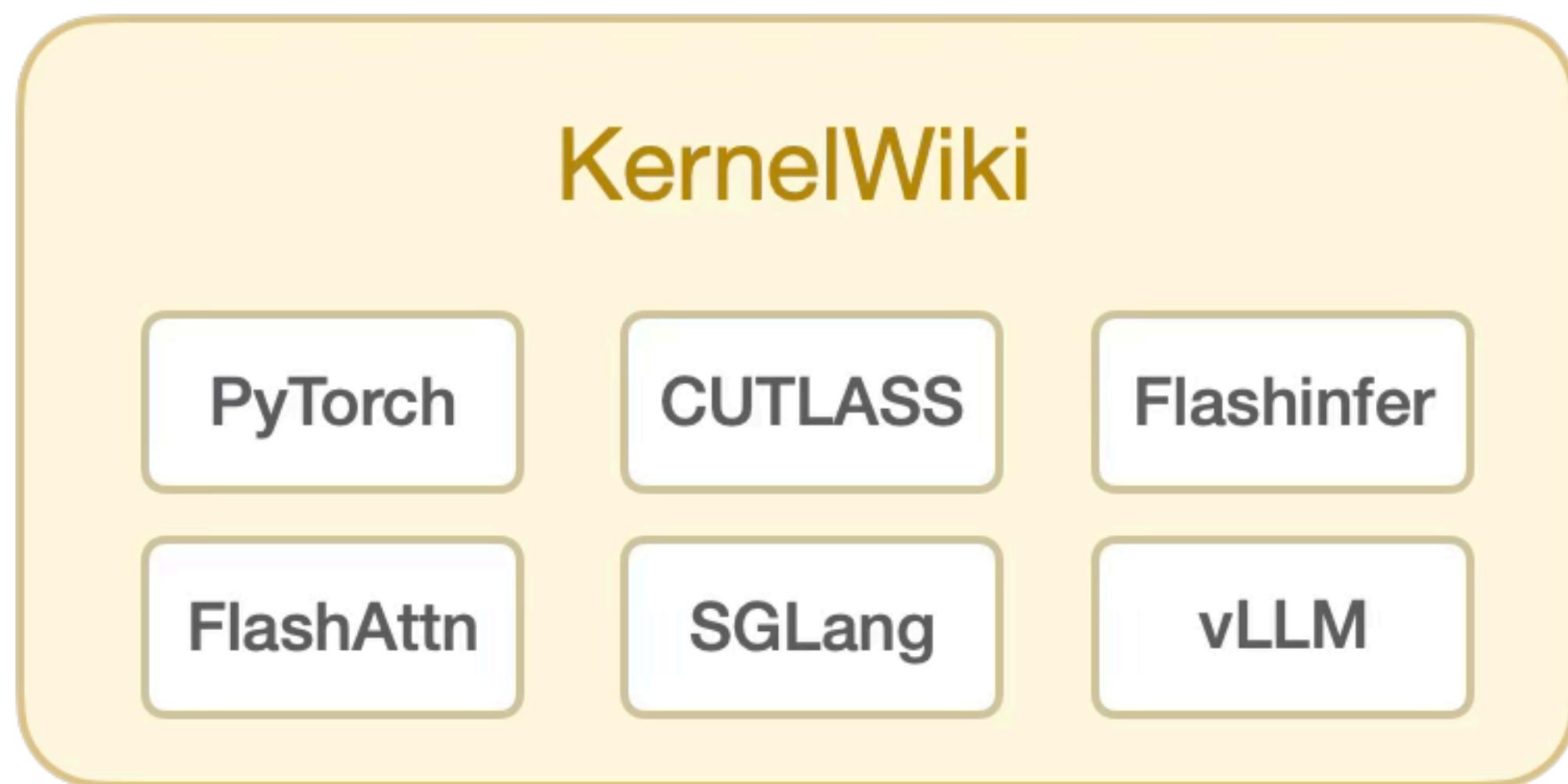
(B) Agent Loop with verification



<https://github.com/PolyArch/humanize>

Kernel Wiki

Domain knowledge as retrievable agent memory



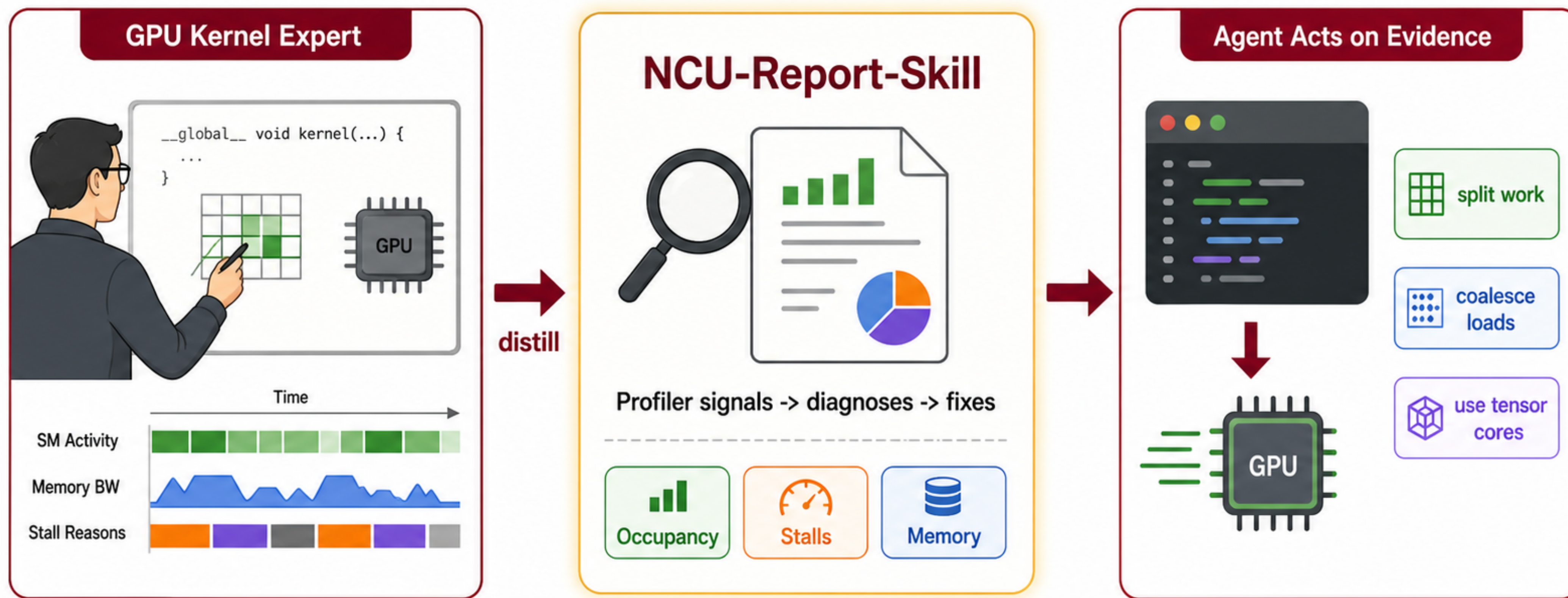
Reference

1. Collect PRs from famous repo
2. Filter by keywords: Blackwell/sm100/moe ...
3. Use Claude to generate label for each PR

SGLang/vLLM/
TensorRT-LLM/PyTorch

NCU-Report-Skill

Distill Expert Profiling Workflow into an Agent Skill

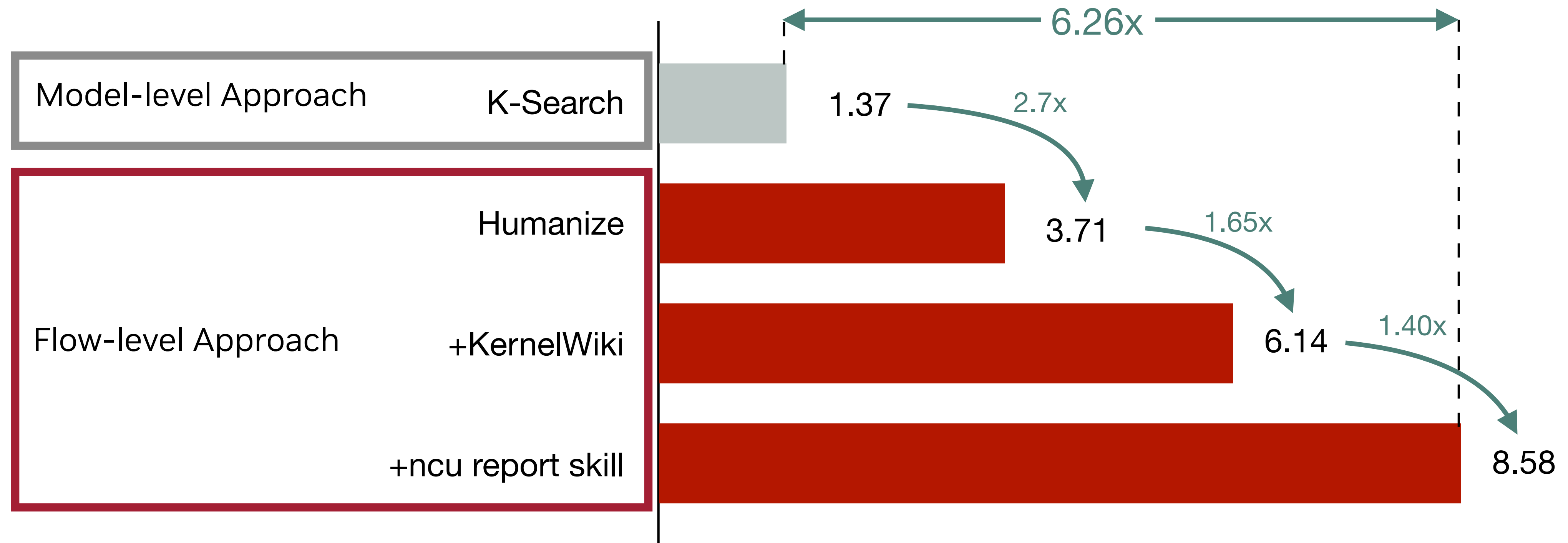


Expert knowledge + profiler evidence = actionable optimization

KDA's General Improvement

Agentic loop w/ little human priors brings the impressive improvements.

DeepSeek Sparse Attention TopK Indexer Kernel



Mean Speedup vs FlashInfer Baseline

KDA's Detailed Analysis on DSA Indexer

- **Official Implementation:** 先用 scalar FP8 score kernel 计算分数，再接 radix-select
 - Bound by score: register pressure 高、occupancy 低，以及 FP8 dot product 没有走到 Blackwell tensor core 路径
- **KDA's Approach:**
 - KernelWiki 在这里改变了搜索方向，它把 Hopper-to-Blackwell migration notes，以及 vLLM、DeepGEMM 等项目中的生产 kernel 经验带进循环：不再沿着 scalar dot-product 继续局部微调，而是把 score computation 重写到 Blackwell 风格的 tensor-core 路径：
 - tcgen05.mma 指令路径；
 - TMA-based paged loads；
 - TMEM-backed accumulation；
 - warp-specialized execution。
 - 重写后 score stage 从 73.6 μ s 降到 7.1 μ s，后续优化才转向 radix-select 和 workload-specific fast path。
- Agent 尝试过 multi-block radix-select，但经过验证发现 synchronization 和 launch overhead 抵消了潜在收益。
 - 最终保留下来的，是一个针对 54% workload ($\text{max_seq_len} \leq \text{topk}$) 的短路径，以及 1024 threads、float4 vectorized loads、2-pass radix、atomicAdd-based fill 等被 benchmark 证明确实有效的 radix-select 优化。
- 提交版本最终达到 19.0758x mean speedup。

KDA's Detailed Analysis on DSA Sparse Attention

- Sparse Attention 的核心难点：对选出的 KV subset 做高效 gather、过滤和 reduction，同时控制稀疏访问带来的访存不连续问题。
- **KDA's 第一版稳定收益来自 split-KV 设计：**
 - 16-way parallelism、ballot-and-popcount sparse filtering、vectorized shared-memory gather 和 log-sum-exp reduction。这个方向利用了真实 sparsity，减少了 official baseline 中的无效计算。
- 随后 KDA 继续探索
 - single-CTA fusion
 - cooperative split-reduce fusion
 - cluster-based launch pattern 等方向，
 - 但多条分支在 verifier-grounded measurement 下表现为中性甚至负收益。最终保留下来的，是更合适的 eager kernel family，以及能够降低 launch overhead 的结构融合。
- 最终，DSA Attention 提交版本相对官方 baseline 达到 4.5425x mean speedup，并通过全部 23 个官方 workload。

KDA's Detailed Analysis on GDN Track - Prefill

- GDN Prefill 的主要有效方向，是为短序列构造 custom CUDA short path：将 state 放在 register 中，并融合 scale、softplus、sigmoid、gating 和 state update，避免 CuTe baseline 在短序列上承担过高的 pipeline startup、TMEM lifecycle、scheduler overhead 和 epilogue cost。
- 在短序列 workload 上，这一路径带来了明确收益：
 - seq_len = 6 约 8.05x speedup；seq_len = 30 约 2.0x speedup；short workload average 约 3.94x。
- 但长序列场景的结论并不相同。Agent 尝试了大约二十个方向，包括 chunk-wise kernel、medium-length scalar extension、2-SM cooperative MMA 和 fusion variants 等，但都没有通过 robust verification。ncu-report-skill 显示，长序列的主要瓶颈是 long-scoreboard memory latency，而不是最初容易假设的 SFU 开销。
- 因此，最终方案采用更克制的 hybrid dispatcher：短序列走 custom CUDA path；+ 长序列回退到 untouched CuTe baseline。
- 这个结果说明，kernel 优化并不总是意味着对所有 shape 强行使用自定义实现。对于部分形态，承认成熟 baseline 更稳，并通过 router 只在合适场景切入 custom path，反而是更优的工程选择。
- 最终，GDN Prefill 提交版本达到 1.9158x arithmetic-mean speedup。

KDA's Detailed Analysis on GDN Track - Decode

- GDN Decode 每次只处理一个 token，kernel 本身已经处在几微秒量级。ncu-report-skill 确实帮助 Agent 找到了 under-vectorized loads 和不理想访存模式等问题，并推动了一些修正；但在这种 regime 下，launch overhead 和数据搬运占比很高，真实 kernel-level 改进也很难转化为端到端大幅收益。提交版本达到官方 baseline 的 0.8023x，属于竞争力接近但未超过 baseline 的结果。

KDA's Detailed Analysis on MoE Track

- 这个 benchmark 来自 DeepSeek-V3/R1 的 FP8 block-scaled mixture-of-experts kernel, 将 routing、FP8 block dequantization、grouped GEMM scheduling 和 weighted accumulation 放在同一个被计分 operator 中。更具挑战的是, 官方 baseline 调用了 closed-source TRT-LLM FP8 GEMM binary, 这意味着可见实现并不能完整解释 baseline 的性能来源。
- KDA 一开始尝试在 CUDA 中复现相关能力, 但没有在比赛时间内收敛。后续方向转向 Triton, 并明确要求 Agent 使用 Triton 3.6 的 Blackwell support: 检查 generated PTX 是否走到 native tcgen05 FP8 path, 再测试 hardware-scaled FP8 GEMM variants。
- 最终, 这个 track 的提交排名是 MoE Track 1st place; 但从官方 baseline 对比看, 提交版本的 mean speedup 是 0.6504x, 并没有超过 baseline。更准确地说, 系统发现了部分有效方向, 却还没有完全复现闭源 baseline 背后的 FP8 GEMM 能力, 距离 human-expert 级别实现仍有差距。
- 值得一提的是, 赛后时间充裕时我们**让 KDA 用 CUDA 实现了比 Baseline 更快的FP8 MoE Kernel (1.31x)**
 - W/ Humanize 2.0 (better Harness)
 - Longer Running Time

KDA - Results

Modern SOTA LLMs are powerful, but to unleash its full power, we need Model X Tool cooperation.

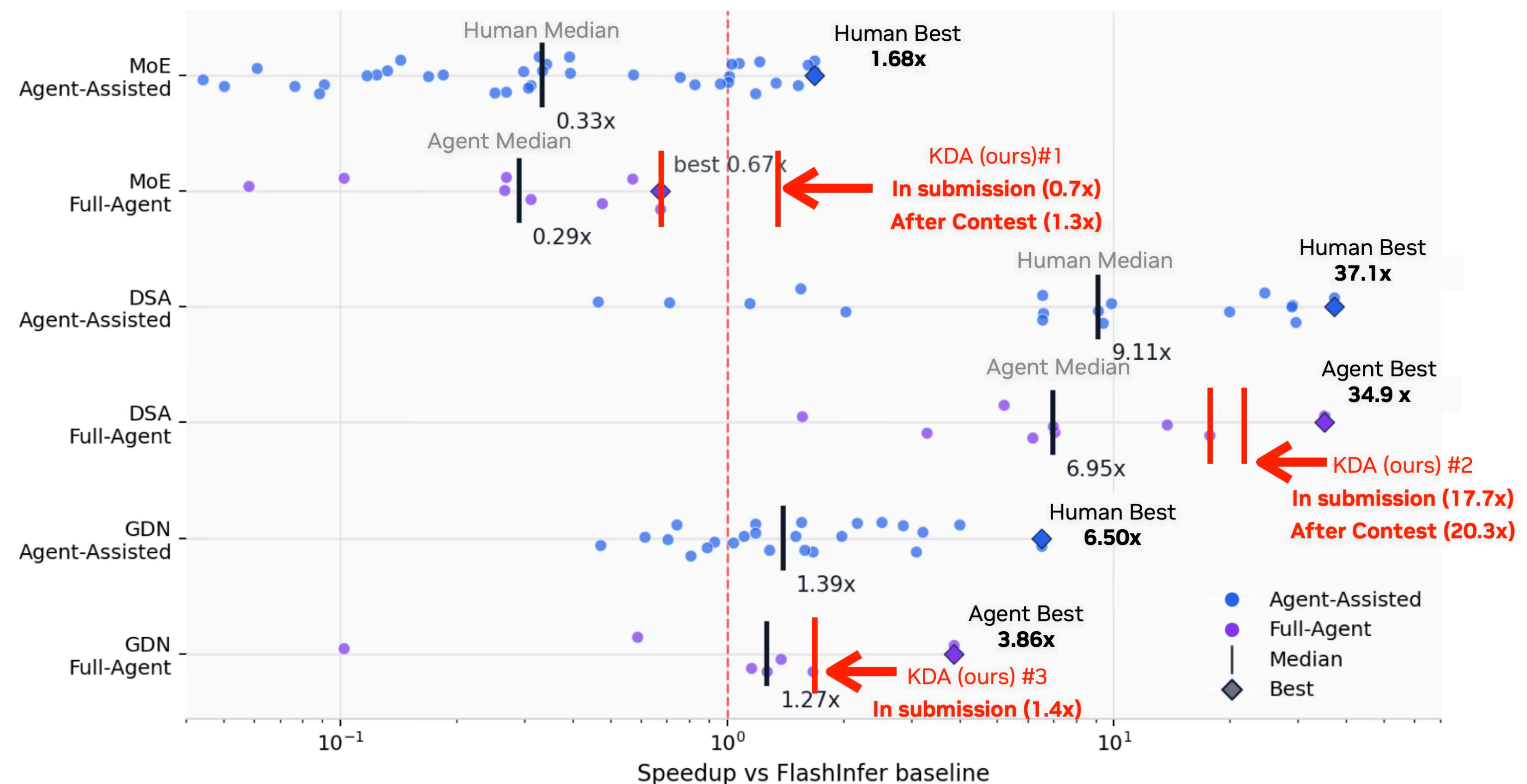
	K-Search	KDA (ours)
DSA	6.15x	15.22x
DSA Top Indexer	1.37x	19.08x
GDN Profile	1.09x	1.92x
GDN Decode	0.99x	0.92x
FP8 MoE	0.27x	1.31x

#1 1st place in the MoE Track — Full-Agent

#2 2nd place in the DSA Track — Full-Agent

#3 3rd place in the GDN Track — Full-Agent

MLSys 2026 FlashInfer Contest Speedup Distribution



Notably: The advisor (**ligeng zhu**) had not written a single line of CUDA in 2026, and the undergrad (**dongyun zou**) had only written about <1k lines of Triton in 2026.

Take-aways

Workflow and domain knowledge matter

- GPU kernels should be vibe-coded: not merely generated by LLM code completion.
- Independent verifiers are essential: they reduce reward hacking and let agents work for 24 hours without human babysitting.
- Domain knowledge is leverage: distill expert workflows into reusable agent skills.





Agents that Explore Open Science Problem

A shown case of Anthropic's Takehome

Ligeng Zhu

Research Scientist, NVIDIA Research

Anthropic's Takehome Challenge

https://github.com/anthropics/original_performance_takehome

- **Goal:** a batched tree-walk computation on a virtual VLIW + SIMD machine.
 - VLIW + SIMD hardware has not been widely produced / deployed, so little leakage in pretraining or post-training.
 - VLIW is hard to program to the pipeline
 - Domain knowledge is not that helpful
- Therefore, anthropic's blog says such challenges is **AI-resistant**
 - <https://www.anthropic.com/engineering/AI-resistant-technical-evaluations>
- **Interview Target:**
 - ❌: how the candidate familiar w/ compute architecture/parallel computation
 - ✅: how fast the candidate could design a "harness" to solve the issue

Anthropic's Takehome Challenge

2164 cycles: Claude Opus 4 after many hours in the test-time compute harness

1790 cycles: Claude Opus 4.5 in a casual Claude Code session, approximately matching the best human performance in 2 hours

1579 cycles: Claude Opus 4.5 after 2 hours in our test-time compute harness

1548 cycles: Claude Sonnet 4.5 after many more than 2 hours of test-time compute

1487 cycles: Claude Opus 4.5 after 11.5 hours in the harness

1363 cycles: Claude Opus 4.5 in an improved test time compute harness after many hours

If you optimize below 1487 cycles, beating Claude's best performance at launch, email us at performance-recruiting@anthropic.com with your code and a resume.

Model-level Agents:

9812 cycles: DeepSeek-V3.2
2901 cycles: Opus 4.6
2784 cycles: Kimi-2.6
2544 cycles: Minimax-2.5
2401 cycles: DeepSeek-V4-Pro

Tool-Level Agents:

1864 cycles: Opus 4.6 oneshot
1755 cycles: Opus 4.7 oneshot
1711 cycles: GPT-5.3-codex
————— (after April 2026) —————
1458 cycles: Claude Code /goal + Opus 4.7
1311 cycles: Codex /goal + GPT-5.5

Flow-Level Agents:

1250 cycles: Humanize 1.0 + GPT-5.4 + Opus 4.6
1001 cycles: Humanize 1.5 + GPT-5.5 + Opus 4.7
994 cycles: Humanize 2.0 + GPT-5.5 + Opus 4.7

- NVR SG: Jincheng Yu 1414 cycles
- Jian Weng (KAUST): 1250 CYCLES
- NVR US: Bian Song 1007 cycles
- NVR: **Zijian Zhang & Ligeng Zhu** 994 cycles (941 w/ some tricks)

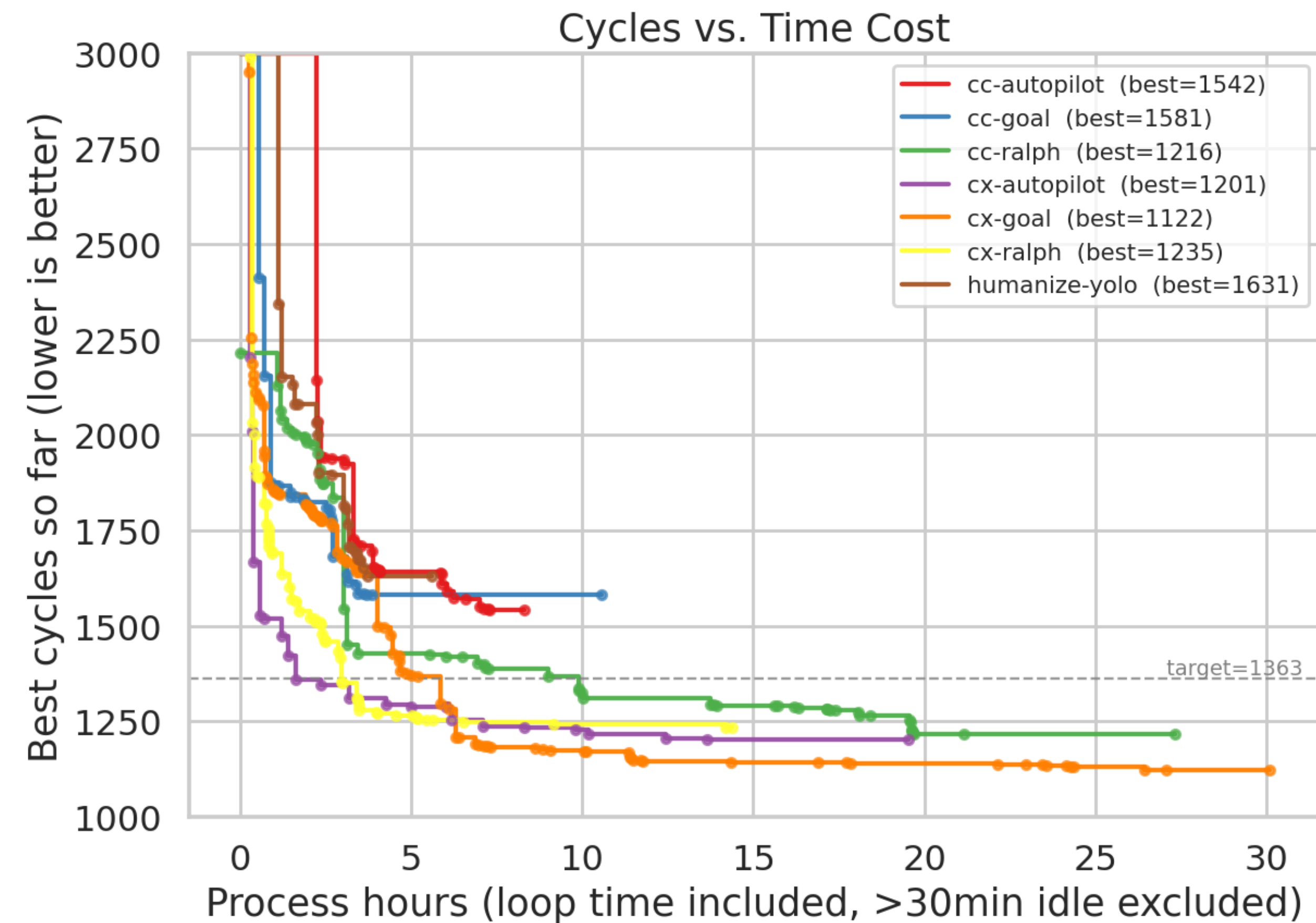
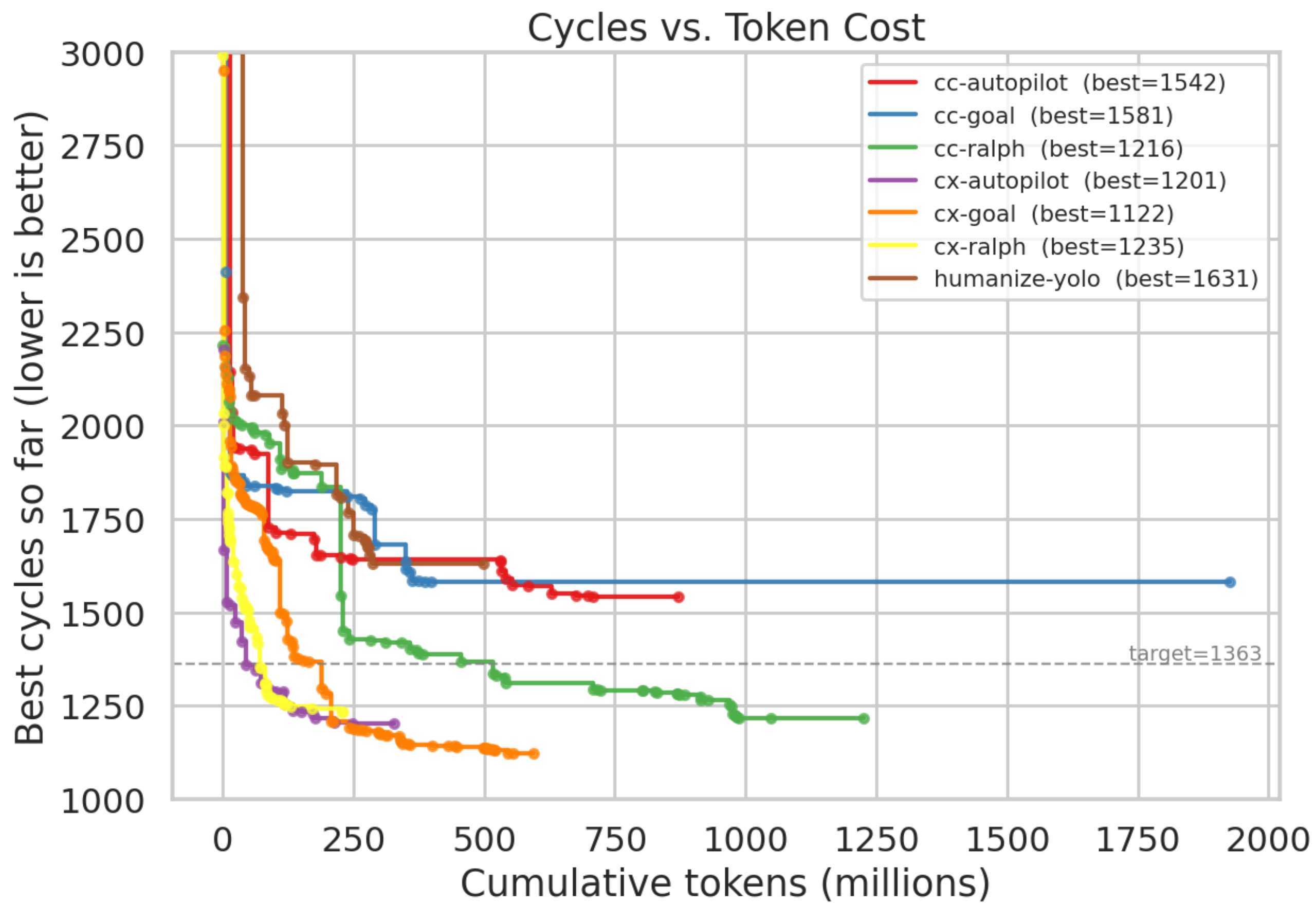
Theoretical limit (read all data) is around ~930 cycles.

994 cycles << 1,487 cycles (a's interview bar)

<https://vliw-challenge.fly.dev/>

“Flow” Matters in Agent Productivity

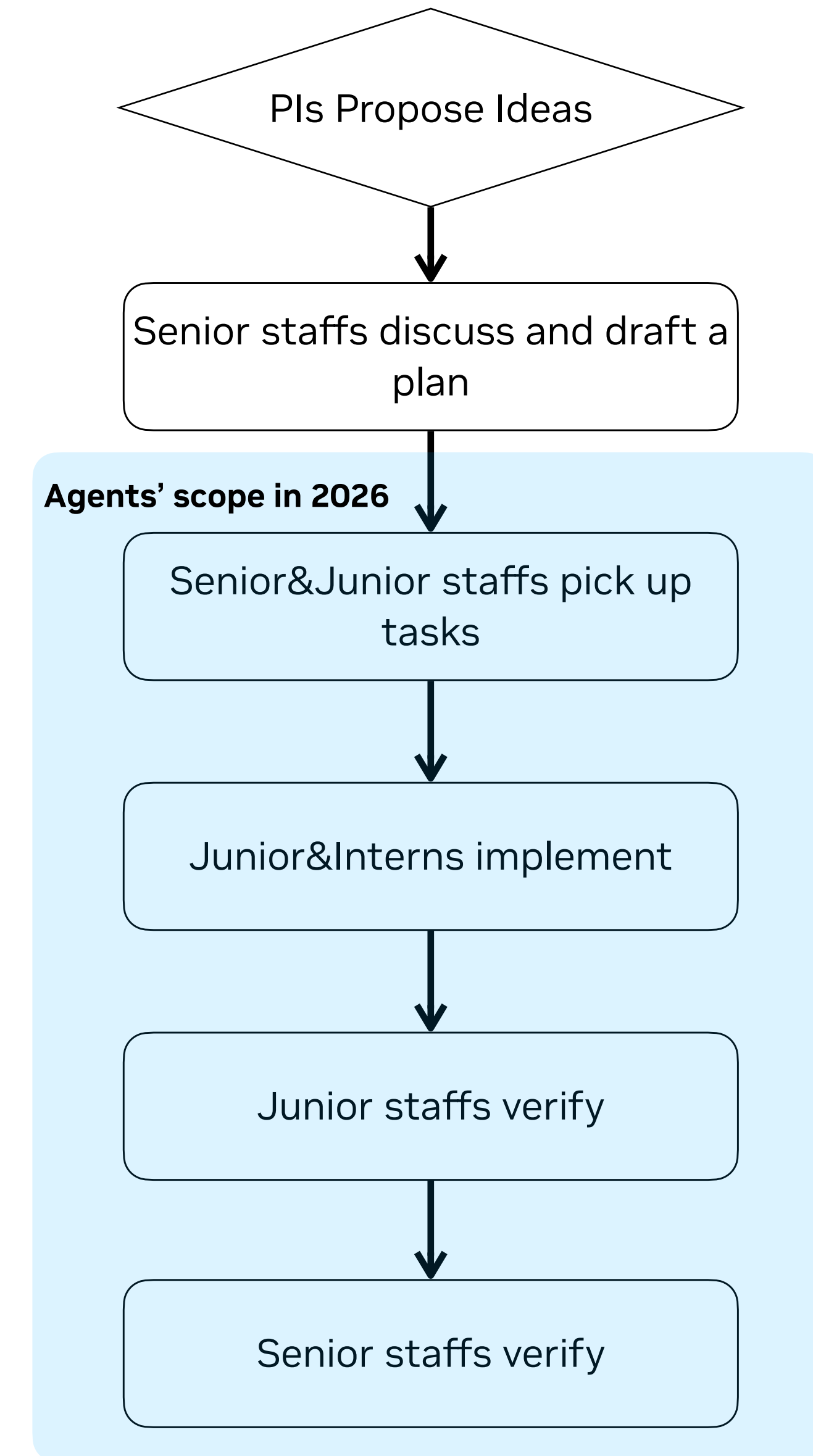
Optimization progress: 7 methods on the perf_takehome kernel



Take-aways

- Vibe Code / Vibe Research =
Model (opus/gpt/deepseek/kimi) +
Tool (claudecode/codex/opencode) +
Flow (humanize/autoresearch/ralph-loop)
- If you miss any piece of the components, your vibe coding experience would suffer greatly!
- When all people use the same model (GPT/Opus) and same tool (Codex/Claude Code), Flow is key to improve the performance of agentic systems.
- Trust your agent as trust your lab-mates
 - Give him the full-access and full-approval
 - Talk to him w/ nature languages as much as possible, rather than interactive coding

$$O(\text{\#tokens}) = O(\text{\#productivity in 2026})$$





Humanize — past, present, and future

From Automated Idea Factory to Realization

Ligeng Zhu

Research Scientist, NVIDIA Research

Humanize — Design Initiatives

- **Belief:**

- LLMs are much more knowledgeable than human (30T token pertaining)
- LLMs remembers more tricks / tips than human (NIAH tests)
- 7x24 agents would eventually outperform human (never tired / rest)



你还能有Codex聪明?

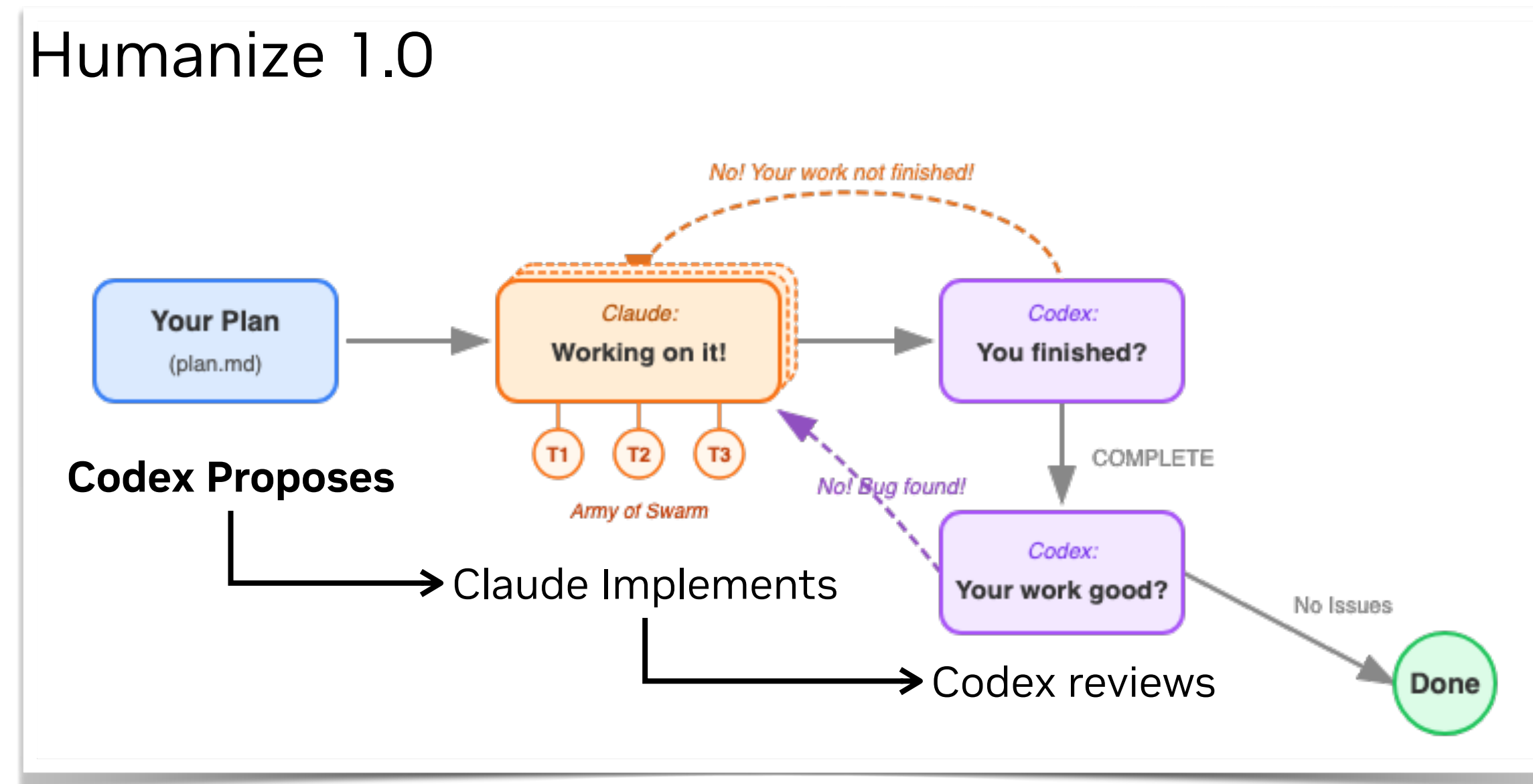
- **OK, but why my agents fails?**



你跟豆包一样聪明!

Humanize — Design Initiatives

- **Agent failure comes from**
 - Mis-aligned initiative — Hugh ambiguity in natural language



- A heave and deep think to ensure the agents and human are on the same page.

Humanize — Design Initiatives

- **Agent failure comes from**
 - Mis-aligned initiative — Hugh ambiguity in natural language

Fast Implementation X
Weak Alignment X
Zero Judgement
= Fast Wrong Garbage Work (**boom!**)

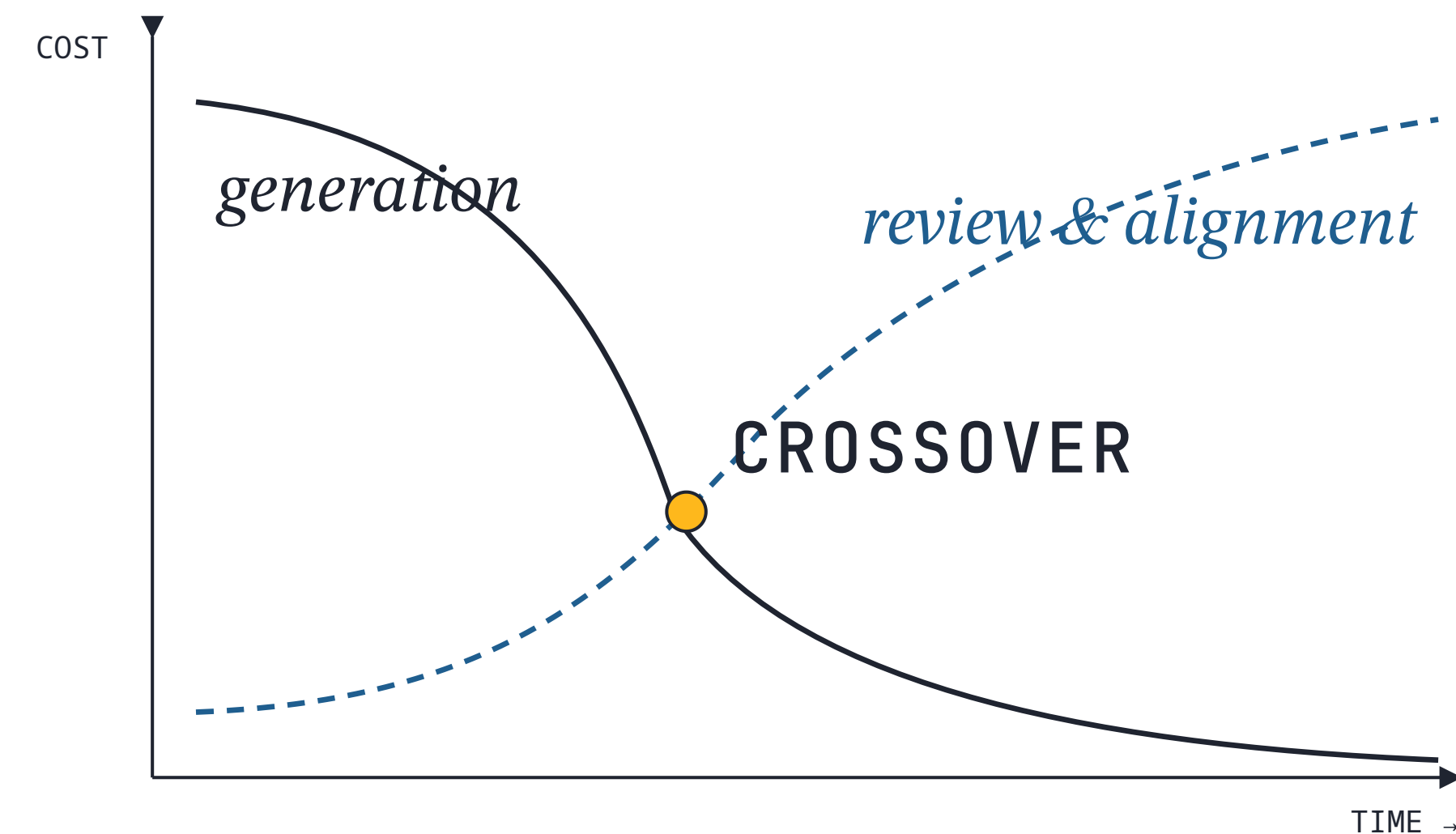
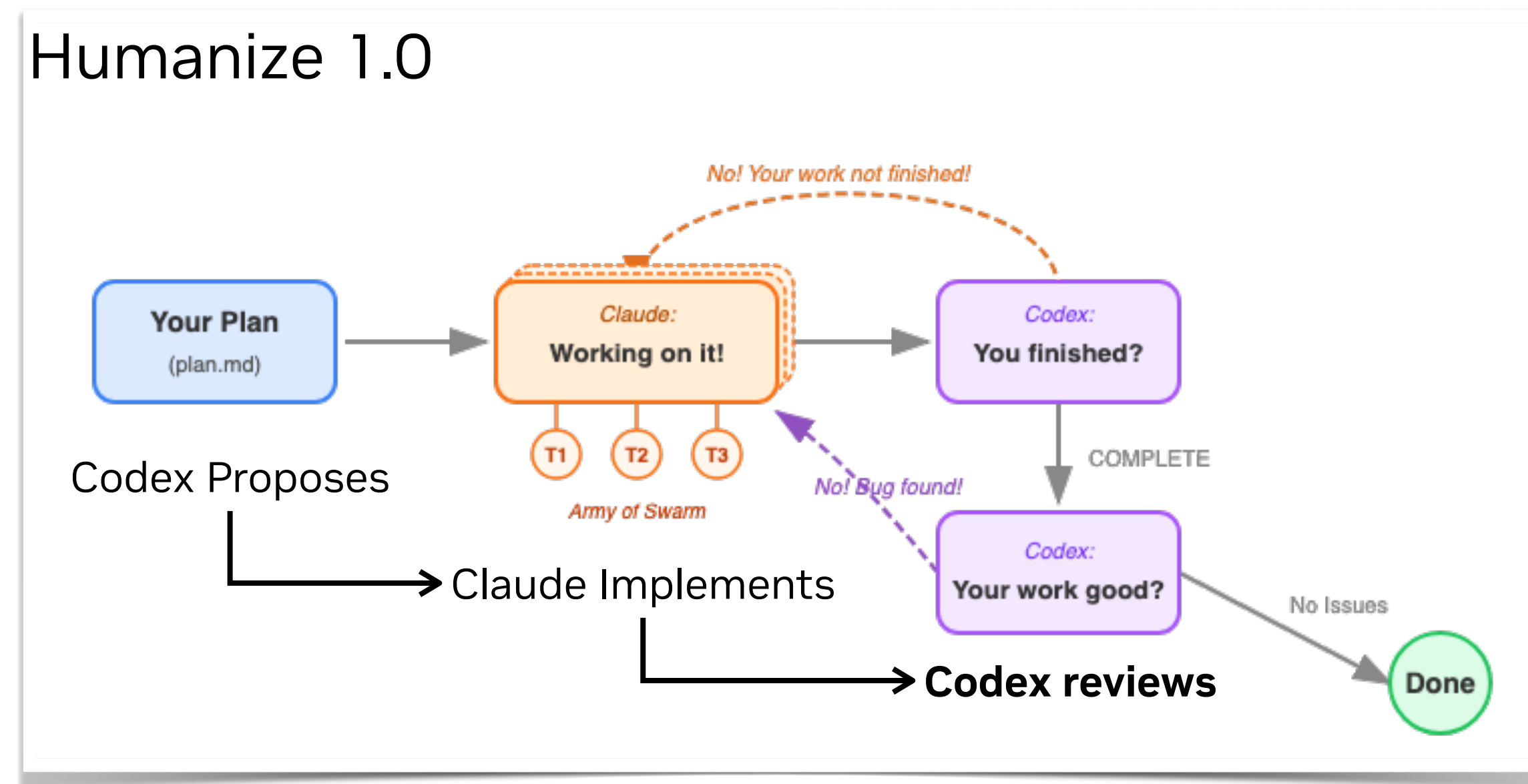


FIG. 1 — BOTTLENECK SHIFT IN AGENTIC CODING WORKFLOWS.

- Humanize 1.0 further **proposes quiz** to check human really understands the initiative

Humanize — Design Initiatives

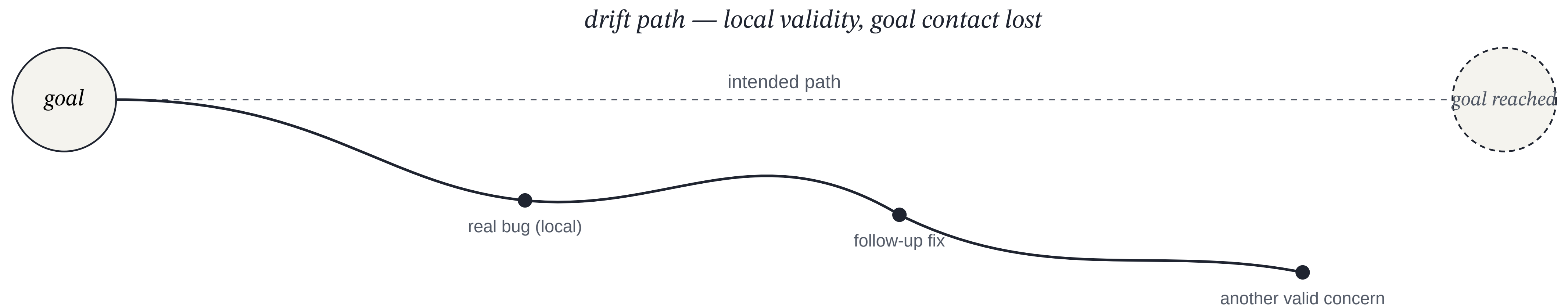
- **Agent failure comes from**
 - Mis-aligned initiative — Hugh ambiguity in natural language
 - Smart = tricky, talent = lazy, easy to satisfy w/ peers



- Must be peer-reviewed to be merged to proceed the next action bullet!

Humanize — Design Initiatives

- **Agent failure comes from**
 - Mis-aligned initiative — Huge ambiguity in natural language
 - Smart = tricky, talent = lazy, easy to satisfy w/ peers



Design response: the progress verdict asks whether the loop still serves the plan, not whether the latest diff is correct. Side issues get a lane, not the steering wheel.

- Peer-Review is crucial to avoid the goal drifting.
- Humanize 1.0 is **highly motivated by the way that human collaborates** on the Github.

Humanize — Design Initiatives

- **From a company / organization angel:**
 - Model ~ = Clever and top new grads
 - Tool ~ = The Cluster / Office Room / Chair / Company Infra
 - Flow ~ = The Proper way of Management and Org Design
- I have a strong belief that **Flow would be the most critical part** of 2026~2027's research.



1:11   87%

✕ Introducing Claude Opus 4.8 \ Anthro ...
www.anthropic.com

AI 

1.0 

Also launching today

In addition to Claude Opus 4.8, we're making the following updates:

- **Dynamic workflows.** This new feature, available in research preview, allows Claude to take on even bigger tasks in Claude Code. Claude can plan the work and then run hundreds of parallel subagents in a single session (and with Opus 4.8, the agents can run for even longer). It then verifies its outputs before reporting back to the user. For example, Claude Code with Opus 4.8 can now carry out codebase-scale migrations across hundreds of thousands of lines of code from kickoff to merge, with the existing test suite as its bar. You can read more about dynamic workflows—available in Claude Code for Enterprise, Team, and Max plans—in [this post](#).
- **Effort control in [claude.ai](#) and Cowork.** A new control alongside the model selector lets users choose how much

IDIA

Humanize — Design Initiatives

- **From a company / organization angel:**
 - Model $\sim$ Clever and top new grads
 - Tool $\sim$ The Cluster / Office Room / Chair / Company Infra
 - Flow $\sim$ The Proper way of Management and Org Design
- I have a strong belief that Flow would be the most critical part of 2026~2027's research.
- Question: Different tasks require the same flow, or different flow?
- **Humanize 2.0: an IR that allows user to write their own flows!**

Humanize — Design Initiatives

- **Humanize 1.0:** A flow that works.
 - A single hard-coded RLCR loop. Builder + Codex reviewer, ralph-loop semantics, severity tags. One way to do one thing — and it does it well.
- **Humanize 2.0:** An IR that builds flows; enabling flows to call flows.
 - Flow primitives in HTML, typed artifacts & boards, a trigger-driven executor, tool-agnostic agents, a local hub for observation & control.

<https://github.com/SihaoLiu/humanize/blob/cdfed2ac28abc712b7158954d4d1d23e2772ea69/docs/spec-aisa.md>

AISA — Agentic Instruction Set Architecture

Version 0.3, post-RISC reduction. AISA is the core intermediate representation of Humanize 2.0. A methodology program is a Development Flow Graph (DFG) whose nodes are AISA instructions and whose edges carry typed bundles of tokens through an ordered dataflow runtime (Kahn Process Networks dialect).

Token	Meaning	Discipline
lcb	Git-worktree handle (Codebase).	LINEAR — single consumer.
lctx	LLM transcript (Context).	LINEAR — single consumer.
lp	Turn-sequence directive (Prompt).	BROADCAST — multi-reader.
lstate	User-declared FSM variant	BROADCAST, scalar, never in bundle

Bundles

A bundle is a tuple shape over the three token kinds {p, cb, ctx}. AISA admits exactly seven non-empty bundles:

<p>	<p,cb>	<p,ctx>
<p,cb,ctx>		
<cb>	<ctx>	<cb,ctx>

```
state @RlcrState { @start Init, Build, Review, Finalize,
                  @end Done, @end Halted, @end Cancelled }
transitions for RlcrState {
  Init    -> Build,
  Build   -> { Review, Halted },
  Review  -> { Build, Finalize, Halted },
  Finalize -> { Done, Build, Halted },
}

agent @Builder = Model(claude_opus_4_7) x Tool(claude_code) x
  Permissions { Edit: Allow, Bash: Ask, SubAgent: Allow }
agent @Reviewer = Model(gpt_5_4) x Tool(codex_cli) x
  Permissions { FileRead: Allow, Review: Allow }

agent @Id = @hmz_stdlib::identity
agent @Final = Model(gpt_5_4) x Tool(codex_cli) x
  Permissions { FileRead: Allow, Review: Allow }

flow @rlcr over RlcrState (%init_cb: <cb>, %init_plan: <p>) -> <cb> {
  %ctx_init = aisa.branch(null) { agent = @hmz_stdlib::ctx_empty, M = 1 }

  // Loop-merge on three carriers
  %cb_iter = aisa.converge(%init_cb, %cb_back)
  { agent = @Id, M = 2, fire_in = ANY }
  %ctx_iter = aisa.converge(%ctx_init, %ctx_back)
  { agent = @Id, M = 2, fire_in = ANY }
  %p_iter = aisa.converge(%init_plan, %p_back)
  { agent = @Id, M = 2, fire_in = ANY }

  // Build phase: 1:1 on bundle <p,cb,ctx>
  %cb_b, %ctx_b, %p_b = aisa.step(%cb_iter, %ctx_iter, %p_iter)
  { agent = @Builder,
    trigger = "implement the plan, write round summary",
    transition = host { next = Review },
    verify = [@AC_BUILD] }

  // Review phase: 1:1 on bundle <p,cb,ctx>, agent-driven state
  %cb_r, %ctx_r, %p_r = aisa.step(%cb_b, %ctx_b, %p_b)
  { agent = @Reviewer,
    trigger = "grade the round summary; emit CONTINUE/COMPLETE/STOP",
    transition = agent {
      candidates = [Build, Finalize, Halted],
      parse = sentinel {
        CONTINUE -> Build,
        COMPLETE -> Finalize,
        STOP -> Halted,
      },
    },
    on_ambiguous = stay,
    max_ambiguity = 2,
  } }
}
```

Humanize 1.0 RLCR Example

From ~700 Loc bash to ~90 Lines

```
// State-driven switch on cb (and on ctx, p in parallel)
%cb_back, %cb_to_final, %cb_halt = aisa.branch(%cb_r)
{ agent = @Id, M = 3, fire_out = ANY,
  select_by_state = [Build, Finalize, Halted] }
%ctx_back, %ctx_to_final, %ctx_halt = aisa.branch(%ctx_r)
{ agent = @Id, M = 3, fire_out = ANY,
  select_by_state = [Build, Finalize, Halted] }
%p_back, %p_to_final, %p_halt = aisa.branch(%p_r)
{ agent = @Id, M = 3, fire_out = ANY,
  select_by_state = [Build, Finalize, Halted] }

// Finalize phase: code review by codex
%cb_final, %ctx_final, %p_final = aisa.step(%cb_to_final, %ctx_to_final, %p_to_final)
{ agent = @Final,
  trigger = "perform full code review; emit Done / Build / Halted",
  transition = agent {
    candidates = [Done, Build, Halted],
    parse = predicate(@hmz_stdlib::codex_review_predicate),
    on_ambiguous = err(InvalidOutput),
  } }

return %cb_final
}
```

AISA — Agentic Instruction Set Architecture

Version 0.3, post-RISC reduction. AISA is the core intermediate representation of Humanize 2.0. A methodology program is a Development Flow Graph (DFG) whose nodes are AISA instructions and whose edges carry typed bundles of tokens through an ordered dataflow runtime (Kahn Process Networks dialect).

Token	Meaning	Discipline
lcb	Git-worktree handle (Codebase).	LINEAR — single consumer.
lctx	LLM transcript (Context).	LINEAR — single consumer.
lp	Turn-sequence directive (Prompt).	BROADCAST — multi-reader.
lstate	User-declared FSM variant	BROADCAST, scalar, never in bundle

Bundles

A bundle is a tuple shape over the three token kinds {p, cb, ctx}. AISA admits exactly seven non-empty bundles:

<p>	<p,cb>	<p,ctx>
<p,cb,ctx>		
<cb>	<ctx>	<cb,ctx>

Humanize 2.0 Preview 5-way Parallel Build-Review

```
flow @parallel_rlcr (%init_cb: <cb>, %mega_plan: <p>) -> <cb> {
  %ctx_empty = aisa.branch(null) { agent = @hmz_stdlib::ctx_empty, M = 1 }

  // Plan split: 1:5 on <p>
  %p1, %p2, %p3, %p4, %p5 = aisa.branch(%mega_plan)
  { agent = @plan_splitter, M = 5, fire_out = ALL }

  // Codebase fork: 1:5 on <cb>
  %cb1, %cb2, %cb3, %cb4, %cb5 = aisa.branch(%init_cb)
  { agent = @hmz_stdlib::worktree_fork, M = 5, fire_out = ALL }

  // Five parallel RLCR lanes, each a nested flow call
  %r1 = aisa.step(%cb1, %ctx_empty, %p1) { agent = @rlcr }
  %r2 = aisa.step(%cb2, %ctx_empty, %p2) { agent = @rlcr }
  %r3 = aisa.step(%cb3, %ctx_empty, %p3) { agent = @rlcr }
  %r4 = aisa.step(%cb4, %ctx_empty, %p4) { agent = @rlcr }
  %r5 = aisa.step(%cb5, %ctx_empty, %p5) { agent = @rlcr }

  // Merge: 5:1 on <cb>
  %final = aisa.converge(%r1, %r2, %r3, %r4, %r5)
  { agent = @merge_coordinator, M = 5, fire_in = ALL,
    trigger = "consolidate 5 candidate branches; keep the strongest" }

  return %final
}
```

Humanize 2.0 Preview 100-Idea Stream + Parallel Evaluate

```
flow @idea_hunt (%cb_seed: <cb>, %seed_prompt: <p>) -> <cb> {
  %ctx_empty = aisa.branch(null) { agent = @hmz_stdlib::ctx_empty, M = 1 }

  // Brainstorm 100 ideas as a 1:100 branch; M is static
  %i1, ..., %i100 = aisa.branch(%seed_prompt)
  { agent = @brainstormer, M = 100, fire_out = ALL,
    trigger = "produce 100 distinct approaches" }

  // Fork the codebase 100 ways
  %cb1, ..., %cb100 = aisa.branch(%cb_seed)
  { agent = @hmz_stdlib::worktree_fork, M = 100, fire_out = ALL }

  // Evaluate each in parallel (compile-time map unroll)
  %e1 = aisa.step(%cb1, %ctx_empty, %i1) { agent = @build_review_profile }
  ... ; %e100 = aisa.step(%cb100, %ctx_empty, %i100) { agent =
    @build_review_profile }

  // Rank and pick best: 100:1 on <cb>
  %best = aisa.converge(%e1, ..., %e100)
  { agent = @ranker, M = 100, fire_in = ALL,
    trigger = "score each candidate, keep the top-scoring branch" }

  return %best
}
```

Humanize — Design Initiatives

- **From a company / organization angel:**
 - Model $\sim$ Clever and top new grads
 - Tool $\sim$ The Cluster / Office Room / Chair / Company Infra
 - Flow $\sim$ The Proper way of Management and Org Design
- **From an IR / Programming Language / Computer Architecture view**
 - Model $\sim$ CPU
 - Tool $\sim$ OS
 - Flow $\sim$ Program (.py .cpp)
 - The new programming language that undergrad students will learn 3~5 years later
 - And, humanize 2.0 wants to be the language

Humanize — Design Initiatives

- **Humanize 1.0:** A flow that works.
 - A single hard-coded RLCR loop. Builder + Codex reviewer, ralph-loop semantics, severity tags. One way to do one thing — and it does it well.
 - **Humanize 2.0:** An IR that builds flows; enabling flows to call flows.
 - Flow primitives in HTML, typed artifacts & boards, a trigger-driven executor, tool-agnostic agents, a local hub for observation & control.
- <https://github.com/SihaoLiu/humanize/blob/cdfed2ac28abc712b7158954d4d1d23e2772ea69/docs/spec-aisa.md>
- **Humanize 3.0:** A platform for flow IRs; targeting flows to generate flows.
 - Once flow is HTML, agents can author flows. The system bootstraps its own coordination logic — h2 is the substrate that makes h3 possible.

H1 • PAST

v1

A flow that works.

A single hard-coded RLCR loop. Builder + Codex reviewer, ralph-loop semantics, severity tags. One way to do one thing — and it does it well.

- Plugin commands
- Bash + worktree files
- Hard-coded loop shape

V 1.17.0 • SHIPPED

H2 • PRESENT

v2

A platform/language humans build flows on.



Flow primitives in HTML, typed artifacts & boards, a trigger-driven executor, tool-agnostic agents, a local hub for observation & control.

- 12 vertex primitives
- Triggered dataflow
- MCP hub + GUI

BRANCH H2-DEV • IN FLIGHT

H3 • FUTURE

v3

Agents create agentic flow.



Once flow is HTML, agents can author flows. The system bootstraps its own coordination logic — h2 is the substrate that makes h3 possible.

- Agents author cartridges
- Self-deploying flows
- Bootstrapped coordination

UNSCHEDULED • SCOPED



JetAutoResearch

A token-efficient harness for agent-driven research

Ligeng Zhu

Research Scientist, NVIDIA Research

JetAutoResearch - Method in One Slide

AutoResearch is cool, can we reduce the cost to leverage more people to use the harness?

Karparthy's [AutoResearch](#) is a mini agent framework that makes an LLM behave like an overnight research intern, but the cost soon grows rapidly (~\$5k token cost per week if running on a 8-GPU node) and soon you will see the page below 😭

Hello,

Your API access has been paused in the **Claude Code** workspace because you exceeded your \$10000.00 monthly spending limit. API access will resume at midnight UTC on May 1, 2026.

Warmly,
The Anthropic Team

JetAutoResearch - Method in One Slide

AutoResearch is cool, can we reduce the cost to leverage more people to use the harness?

program.md

LOOP FOREVER:

1. Look at the git state: the current branch/commit we're on
2. **Tune `train.py` with an experimental idea by directly hacking the code.**
3. git commit
4. Run the experiment: `uv run train.py > run.log 2>&1` (redirect everything — do NOT use tee or let output flood your context)
5. Read out the results: `grep "^val\_bpb:|^peak\_vram\_mb:" run.log`
6. If the grep output is empty, the run crashed. Run `tail -n 50 run.log` to read the Python stack trace and attempt a fix. If you can't get things to work after more than a few attempts, give up.
7. Record the results in the tsv (NOTE: do not commit the results.tsv file, leave it untracked by git)
8. If val_bpb improved (lower), you "advance" the branch, keeping the git commit
9. If val_bpb is equal or worse, you git reset back to where you started

Two types of workloads:

1. the high-value reasoning for creative tasks that benefit from the model's intelligence, such as proposing code changes, diagnosing crashes, and suggesting new directions.
2. the low-value orchestration for the workflow (65% tokens) that manages state transitions, retries, and branch advancement or rollback.

Main Idea:

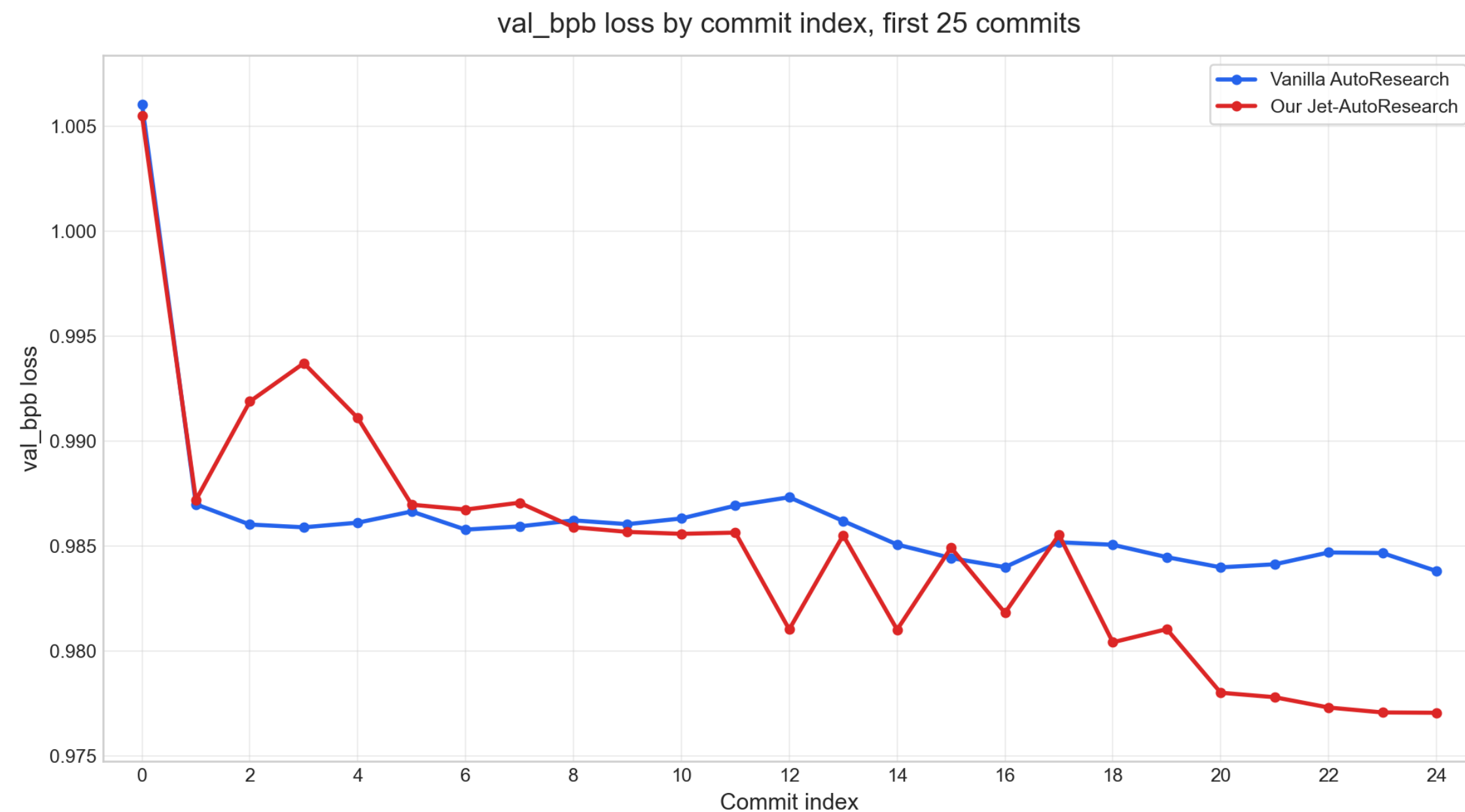
“Compile” the low-value orchestration from LLM execution to python/
bash scripts.

JetAutoResearch - Results

Same Results, but much less tokens.

The same optimization convergence speed, but much less cost (on GPT5.4 xhigh fast)

Easy to use as an ahead-of-time compiler for harness



```
# vanilla autoresearch
codex "launch the @program.md"

# jet-autoresearch
codex "$jet-compile-plan program.md program.agentify.py"
codex "python program.agentify.py"
```

1. Average run cost drops from about `\$17.1` to `\$7.6` (**54% lower**).
2. Average decode tokens drop from `24,251` to `4,593` (**71% lower**).

“

Thanks, questions?

”