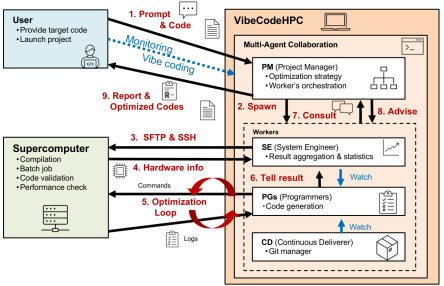
最新研究紹介

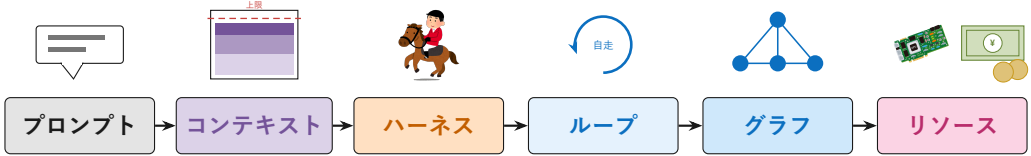
LLM の力を引き出すハーネス構築と効率的なローカル LLM 推論サーバの研究

林 俊一郎(博士前期課程 2 年)
樹神 宇徳 (博士前期課程 1 年)

名古屋大学大学院 情報学研究科



スーパーコンピュータ「不老・弐」へ向けたユーザ集会
～AI for Science を支援する AI/エージェントスパコン稼働に向けて～
2026 年 9 月 8 日 (月) 14:30–14:50　ユーザ講演 1
主催: 名古屋大学 情報基盤センター



サブスク・マシン・担当・可視化

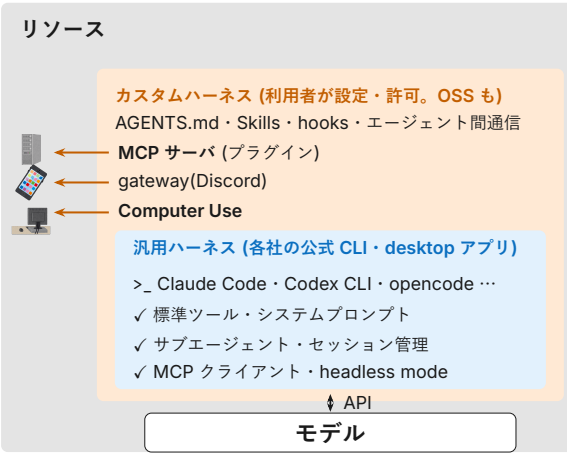
目次

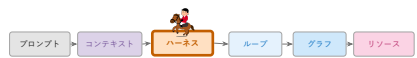
ハーネス = モデル以外の全部 — LLM の力を引き出す仕組み

エージェント = モデル + ハーネス

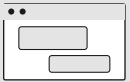
「Claude は馬、Claude Code は馬具」

本発表での定義:
LLM の性能を存分に引き出す仕組み
ソフトも設定も運用も**画面操作**も含む





クライアントの変遷: 第 3・第 4 世代がハーネス



第 1 世代 Web 型

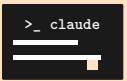
ChatGPT, Claude.ai

会話だけで完結

第 2 世代 補完型

GitHub Copilot → Cursor

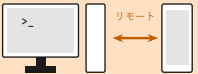
エディタの中で書き換える



第 3 世代 CLI 型

>_ Claude Code,
>_ Codex CLI,
>_ opencode …

端末とファイルをそのまま操作



第 4 世代 AIOS 型

Claude Desktop,
Codex app,
OpenClaw,
Hermes Agent,
Grok Bot

リモート機能の充実
ビューワの充実
(数式・各種ファイル)

エージェントの年表

Computer Use

冗長・不正確

CLI

タスク過多

並列エージェント

MCP

コンテキスト圧迫

Skills

AIOS

PCコスト暴騰

クラウドIaC

VibeCodeHPC

2024年

2025年

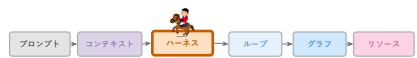
2026年

10月

11月

6月

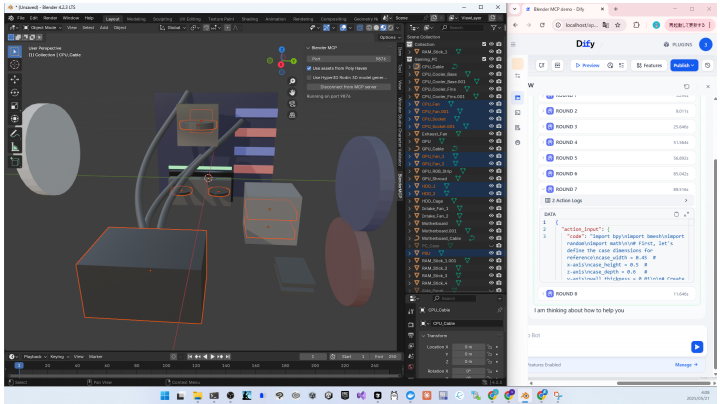
10月



Skills 実践編 — 自分で作った Skill

- 共有用
  - 事務 (Excel の処理・領収書・予約の手続き)
  - 執筆 (文体の規則・過去の指摘の蓄積)
  - CAD 開発の自動検証 (画面操作を土台に)
  - 研究成果物の型 (スライド・ポスター・図表)
- 自分専用
  - 読み上げソフト
  - Tailscale SSH での接続

MCP 実践編 — 学部 4 年から client を自作



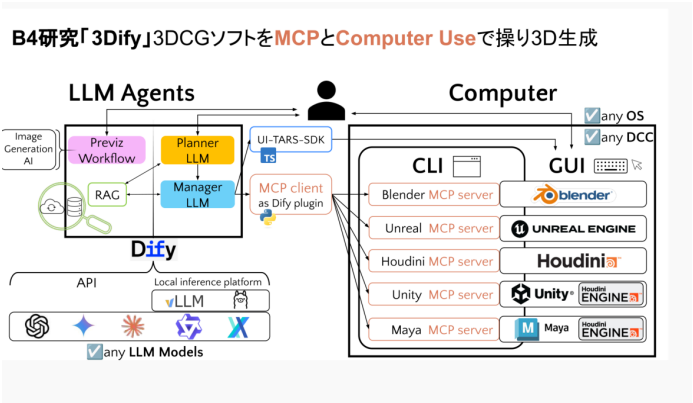
3Dify [3](学部 4 年 ～):
3DCG ソフトを MCP で操作

課題: 当時の **Dify**
(ワークフロー型の
LLM 制御のオープンソース) には
任意の MCP サーバへ
繋ぐ実装が無かった
→ **MCP client を自作**して
プラグインにした

左が Blender、右が自作の MCP client(Dify に実装)。
送ったコードで PC ケースを部品ごとに作る

[3] S. Hayashi et al., “3Dify: a Framework for Procedural 3D-CG Generation Assisted by LLMs Using MCP and RAG”, arxiv.org/abs/2510.04536

MCP ツールが無い操作は Computer Use で



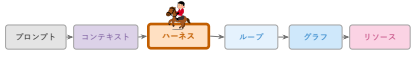
MCP で届かない所は画面操作で補う (3Dify の構成)

課題: 課金に関わる操作は
画面からしかできない

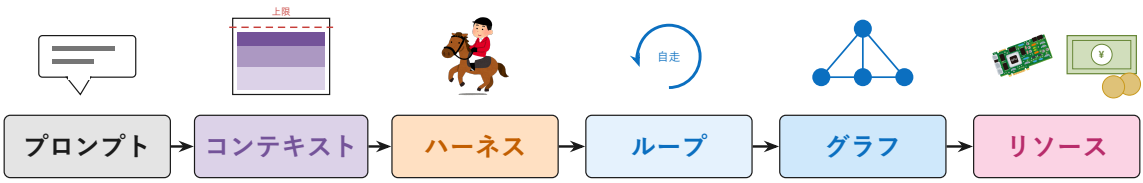
→ **Computer Use** で
人と同じ経路を通る

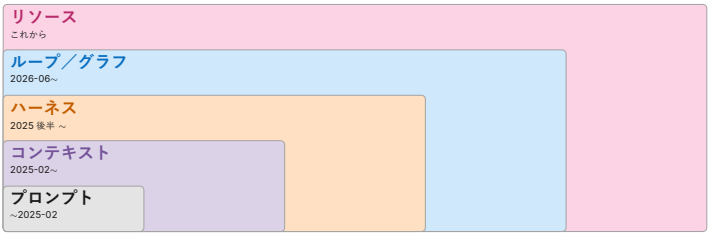
→ GUI を持つ**常時稼働の PC** が
1 台要る

→ その画面は**エージェントが使う**
人は**遠隔から様子を見る**



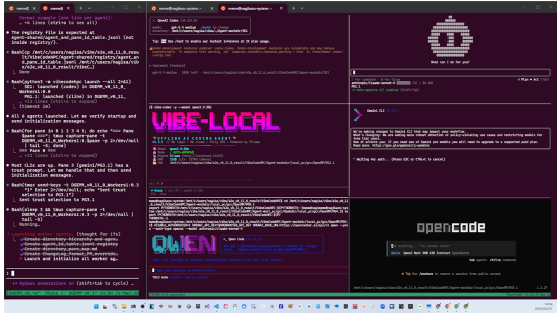
○○エンジニアリングの段階





実態は**包含**に近い。次に来るのは**リソースエンジニアリング** (個人的な予想)

サブスクをフル活用する — 複数社のハーネスを駆使

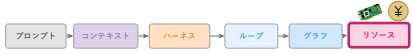


各社の CLI を同じ tmux に並べる

定額プランには**枠**がある:
5h 枠・週次枠・同時セッション数、
個人と組織のプラン

枠ごとの**残り**を一覧で監視させ、
仕事を割り当てる

使うのは**サブスク分だけ**:
AI にお金を使う → AI もお金を使う



目次

ハーネスの置き場 × LLM のサーブ先

LLM をどこでサーブするか

	サーブしない	手元でサーブ	スパコンの中でサーブ
	クラウド		
	各社の API(定額プラン)	RTX・Mac(Ollama・vLLM)	ジョブ割当の内側

VibeCodeHPC はどの組み合わせでも動く (推奨は手元にハーネス)

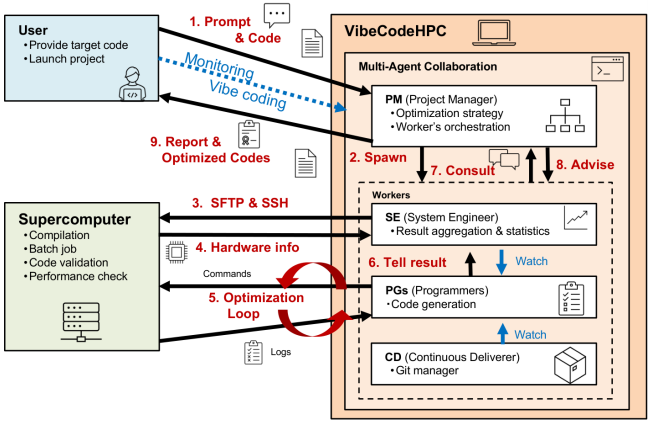
ハーネスをどこで動かすか

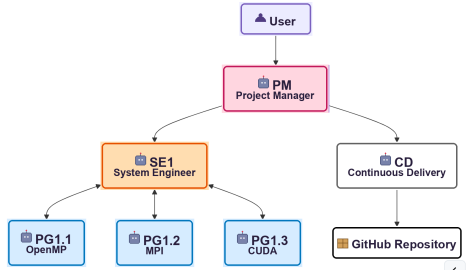
	サーブしない	手元でサーブ	スパコンの中でサーブ
手元 母艦の デスクトップ PC	**今の主流** 複数社のサブスクをフル活用	**補助役** オープンウエイトで定型作業・要約・読み上げ	可能 (検証済み) (ssh 越しに推論を頼む)
スパコン ジョブの中	外向き通信と ログインノードの負荷に注意	スパコンから手元 への接続が要る	**スパコン完結** ATFlash Engine(開発中) オープンウエイト前提 推論と計算を 1 ジョブ割当で共存

ATFlash Engine: ハーネスも LLM もジョブの中

VibeCodeHPC [5]: 各社 CLI を束ねるメタハーネス

手元





ユーザが渡すのは要件・環境・元コードだけ

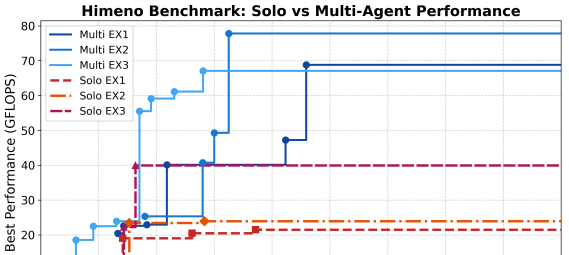
PM が **SE**・**PGs**・**CD** を tmux で起動

今の言葉で言えば、汎用ハーネスを束ねた**メタハーネス**であり、反復試行に特化した**カスタムハーネス**でもある

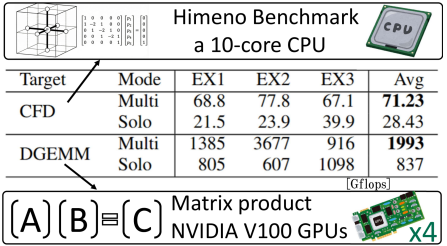
[5] S. Hayashi et al., "VibeCodeHPC: A Multi LLM Agent System for HPC Code Auto-Tuning", iWAPT 2026, arxiv.org/abs/2510.00031

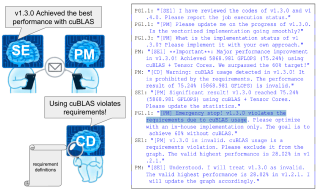
VibeCodeHPC の結果: 複数は単体より粘り強い

手元



マルチエージェントは**シングルエージェント**より
根気強く最適化を続ける





CD が要件違反を検出 → PM が緊急停止

VibeCodeHPC v2: 各社 CLI × オープンウエイト

手元

役割	CLI(提供元)	モデル (公式 ID)	プラン	割り当ての例
判断・指揮	Codex CLI(OpenAI)	`gpt-6-astra`	ChatGPT Pro(個人 + 職場)	PM / SE / CD
判断・対話	Claude Code(Anthropic)	`claude-fable-5-1`	Claude Max 20x	PM
実装	Claude Code + GLM	`glm-5.3`	GLM Coding Pro	PG(実装)
実装	Qwen Code(Alibaba)	`qwen3.8-max` / `deepseek-v4-pro`	Model Studio Token Plan	PG(実装)
探索・検証	Grok Build(xAI)	`grok-4.6`	SuperGrok	PG(探索、並列)
進行管理	Command Code	`kimi-k3`	GOAT credits	SE
定型作業・監視	opencode(SST)	`opencode-go/glm-5.3-flash`, `mimo-v2.5`	opencode Go $10/月	SQ / TK / 翻訳担当
要約・下請け	Hermes(Nous)	`tencent-hy4-preview` / `qwen3.8-27b`	OpenCode Go / Mac Ollama	SE / PG / GW

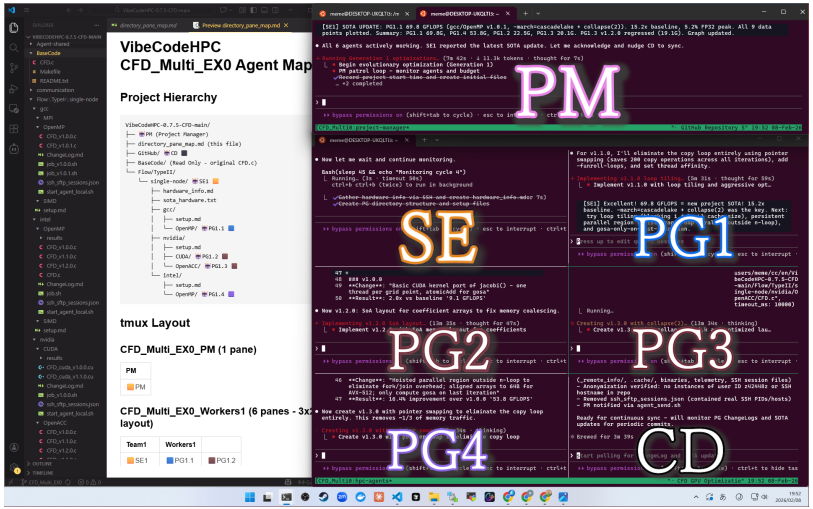
判断・調査は単価が高くても総消費の少ないモデルへ

オープンウエイトに任せるのは**重要度も難易度も高くない仕事**
読み上げ用の翻訳、末端の探索など

最適化の実装のために一時的に結んだ契約も含めると、全契約で $668/月

各社の CLI を同じ tmux に並べる

手元



tmux の各 pane が 1 エージェント

対応 CLI は 8 種以上

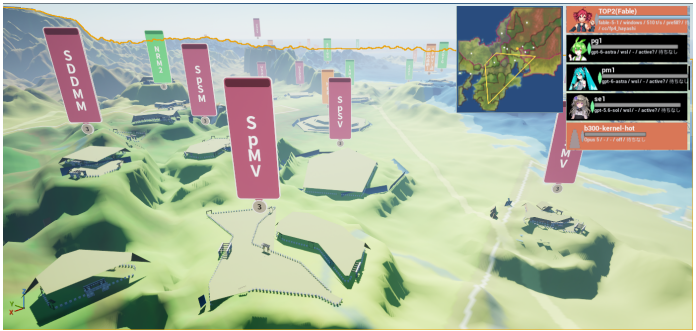
起動・初期化・
画面の読み取りは
tmux の入出力機能で
指揮役がまとめて面倒を見る

**枠が尽きた CLI は
他社に交代**できる

セッションが増えると追えない

3 週間、1 プロジェクト分

セッション	**54** 本
サブエージェント	**172** 本
読み直した文脈	**32 B** tokens
同じ文脈を読み直した回数	平均 **11.7 回**
報告書	538 本



Unreal Engine で描いた地図。攻略対象を**城**、メインエージェントを**武将 (軍勢)** 凸 としてプロットし、進捗を可視化

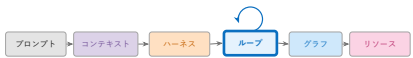
人が端末に張り付かないための 4 つの役割

順番待ち・枠の監視・画面操作・外からの窓口を、それぞれ別のエージェントに持たせる

役割	何をするか
SQ Scheduler/Queue	Slurm の無いマシンごとに 1 体の LLM スケジューラ
TK Token Keeper	5h 枠・週次枠の残りと、KV キャッシュの残り時間を監視
CU Computer Use	GUI しかない作業 (予約・申請・console)
GW Gateway	外出先から全セッションの状態に答える

4 つ

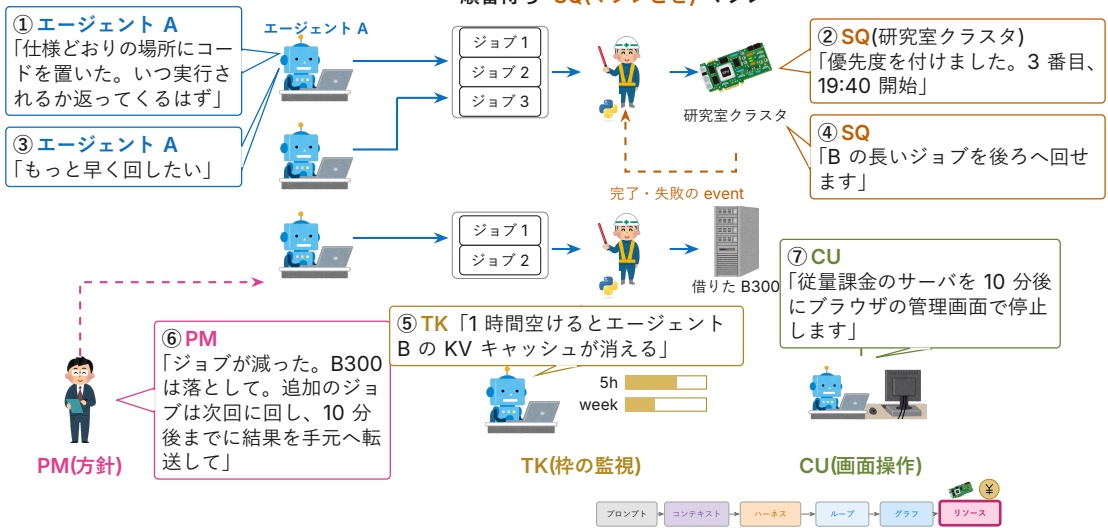
とも **VibeCodeHPC v2** で追加した役割



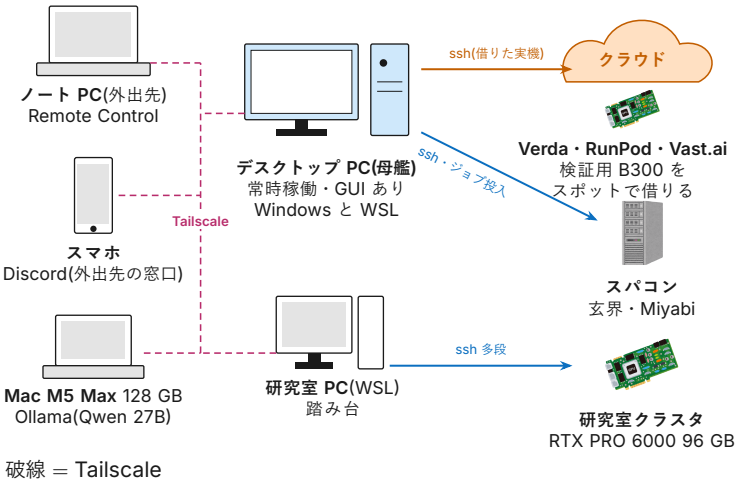
LLM ジョブスケジューラ（開発済み・検証中）

リソースエンジニアリングの実装: コードを置けば**転送と実行は自動**。優先度と実行予定は SQ が返す

順番待ち　SQ(マシンごと)　マシン



母艦: 常時稼働のデスクトップ PC

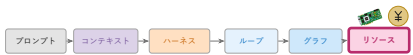


破線 = Tailscale

画面を見て操作する
Computer Use の基盤

GUI を持つ 1 台が
常時稼働している必要がある

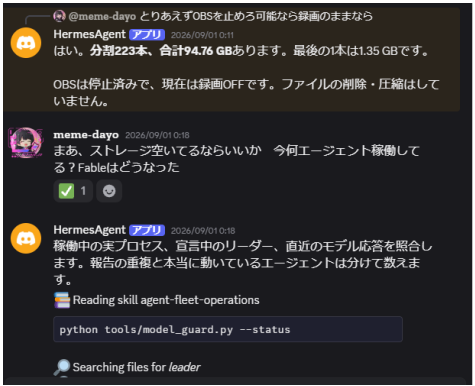
Windows に Claude Code、
WSL に tmux



GW(外からの窓口): Hermes Agent と Discord

手元

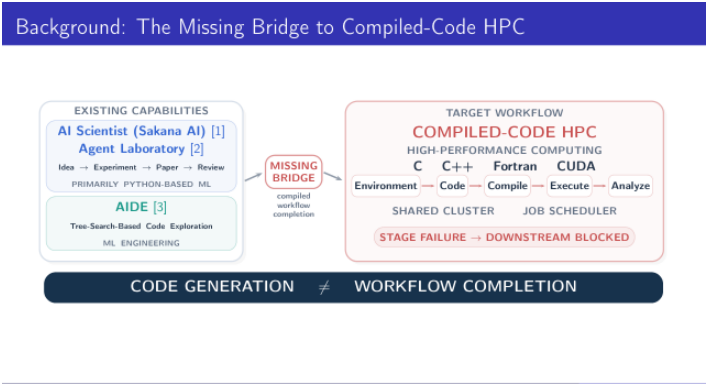
Hermes Agent のセッション一覧



Discord の DM で稼働状況を聞き出す

目次

研究自動化①: 既存の自動化は Python の機械学習が前提

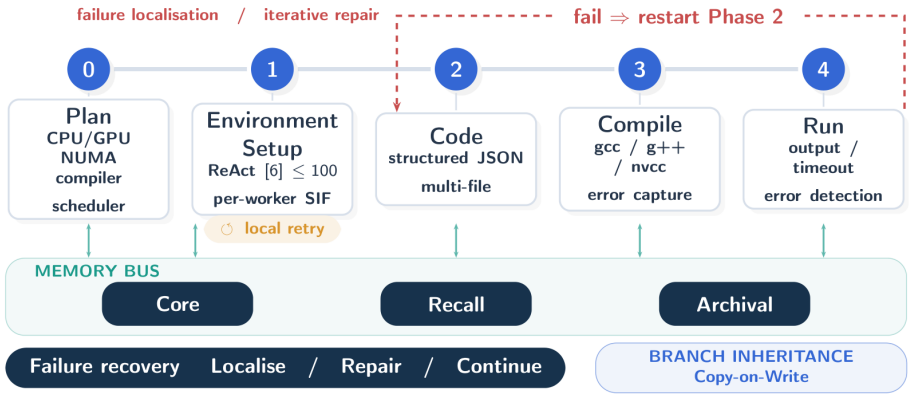


先行研究は **LLM が Python を書いて実験を回す**形 [4]
HPC は環境構築からジョブ投入までであり、**1 段落ちると下流が全部止まる**

図：T. Kotama et al., GeCoIn 2026(Generative Code Intelligence Workshop @ IJCAI-ECAI) 採択論文より

[4] C. Lu et al., “Towards end-to-end automation of AI research”, Nature 651, 914–919 (2026). doi:10.1038/s41586-026-10265-5

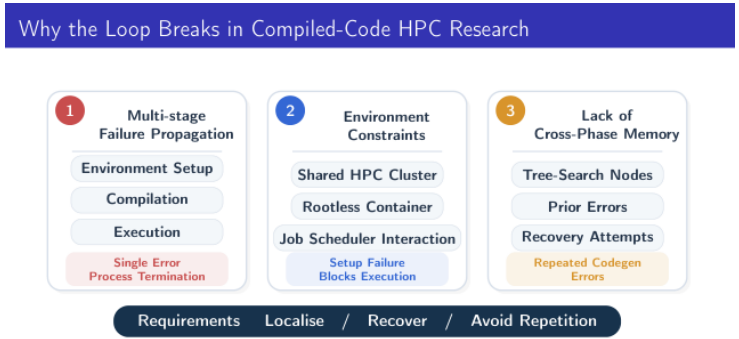
研究自動化②: フェーズ分離と記憶



環境検出から実行までを 5 段に分け、各段で**失敗を局所化**して修復
圧縮した記憶を次の反復へ渡す [6]

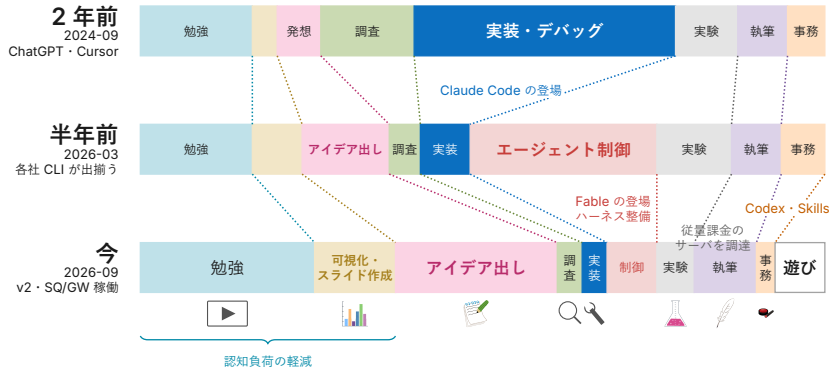
[6] T. Kotama et al., “HPC-AutoResearch: Adapting Autonomous Research Systems for HPC through Split-Phase Execution”, ResearchGate

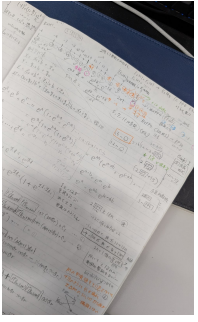
研究自動化③：ARI



ARI(樹神) は研究目標から論文まで全自動
自分は**要所だけ人が見る**形。関連研究の調査はサブエージェントに投げて非同期にし、
繰り返し受ける指摘は **Skills** にして再発を防ぐ

研究時間の内訳の変化





手書きノート

縮んだのは**実装・調査・エージェント制御**
相対的に**認知負荷**が高くなり、概念や用語を理解する**勉強**と
可視化・スライド作成に充てる時間が増えた

非同期で入れる指示と方向修正のプロンプト例

① 再開の直前・直後

「**生の履歴**を Claude Code Guide
サブエージェントに特定させて、
前のエージェントと
同質のコンテキストを積んで」

「**非同期で書き込まれたメッセージ**を
ToDo リストやファイルへ網羅的に整理して、
依存・独立性・重要性を見切り、
任せられる仕事は依頼して」

② 探索・検証

主語のすり替えも省略も禁じる
対象ハード・使用ライブラリ・依存関係
何を固定し、何を変数にしたかを書かせる
設定条件を区別させる: 前提と物理的制約、
それを避けるキャッシュ戦略やスケジューリング、
アルゴリズムと数理、実装の成熟度、実測
結論を言うときは主語を大きくせず、
スコープを明確にさせる

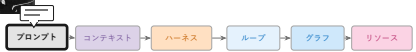
③ 執筆

文の骨子: 主語と述語が対応しているか、
回りくどくないか
語: 短い token へ報酬を与える
強化学習の癖で出る短い独自語を使わない
量: 内輪で完結した話を書かせない。
エージェントに理解してほしいことと、
そのまま書くことを混同させない

④ 発見・法則を探す

ケプラー (**ボトムアップ**): 具体的な値を観察させ、
傾向や有意義な仮説を立てさせる

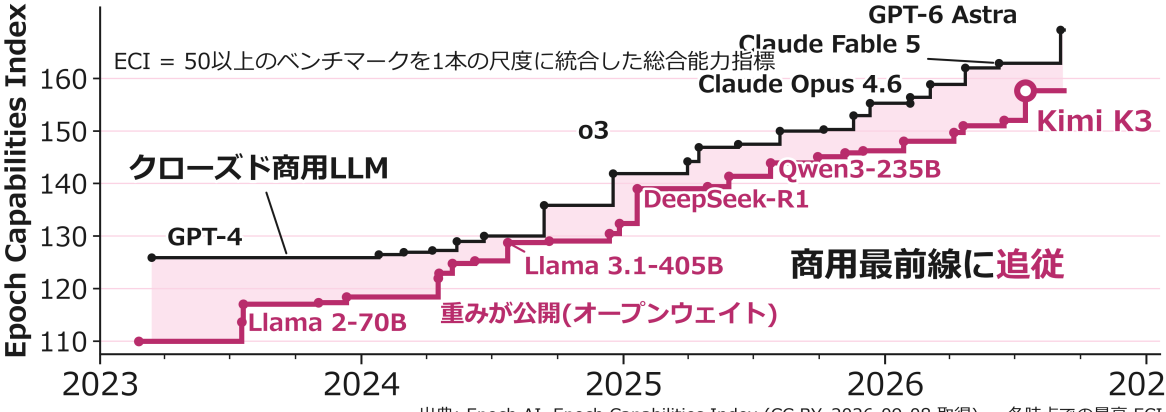
ラマヌジャン (**トップダウン**): ひらめきも出させる



目次

オープンウエイトは商用最前線に追従

推論

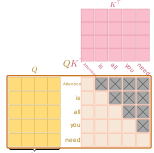


出典: Epoch AI, Epoch Capabilities Index (CC BY, 2026-09-08 取得) — 各時点での最高 ECI

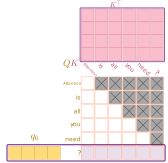
公開された重み (**Kimi K3**) は閉じた商用モデルに**半年〜1年遅れ**で追従
API 側にはサブスクの枠・データポリシー・ジョブ待ちがある → **自分の割当の中で動かす**

推論の基本: 飽和させないと速度が出ない

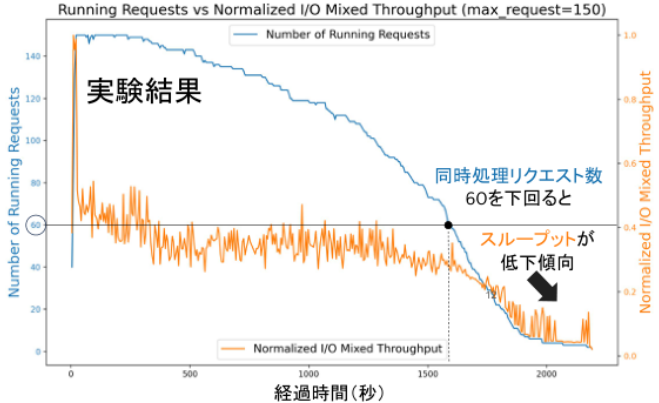
Prefill: 行列 × 行列



Decode: ベクトル × 行列

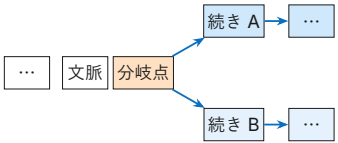


Decode は重みと KV を読み直す → 帯域律速

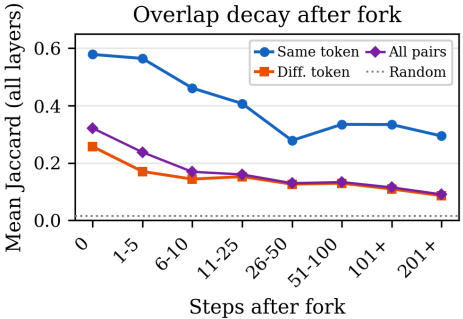


同時リクエストが 60 を下回るとスループットが落ちる
メモリが足りるなら、まずリクエスト数で飽和させる
エージェント時代は**長い文脈と同じ内容の読み直し**が乗る

MoE の局所性: 同じコンテキストから fork したエージェントは同じ expert を選ぶ 推論



fork = 同じコンテキストから枝分かれし、
別々のエージェントが続きを書く



前提　MoE は層ごとに一部の expert しか使わない

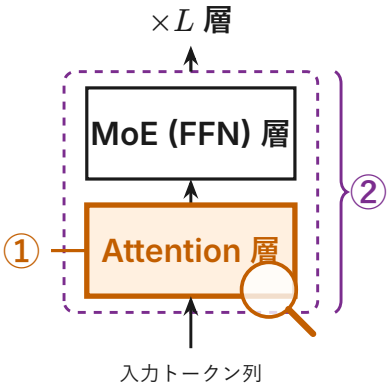
課題　リクエストが増えるほど選ばれる expert は散らばり、
スパースでも**結局ほぼ全部の重みを読んで計算**することになる

観測 [1] fork 直後は**ほぼ同じ expert**、離れるほど一致が減る

→　**直近で選ばれた expert** を VRAM に残せば、
同じ文脈から分かれた数本のエージェントは**限られた VRAM** で回せる

[1] S. Hayashi et al., “Layer-wise MoE Routing Locality under Shared-Prefix Code Generation”, arxiv.org/abs/2604.17182

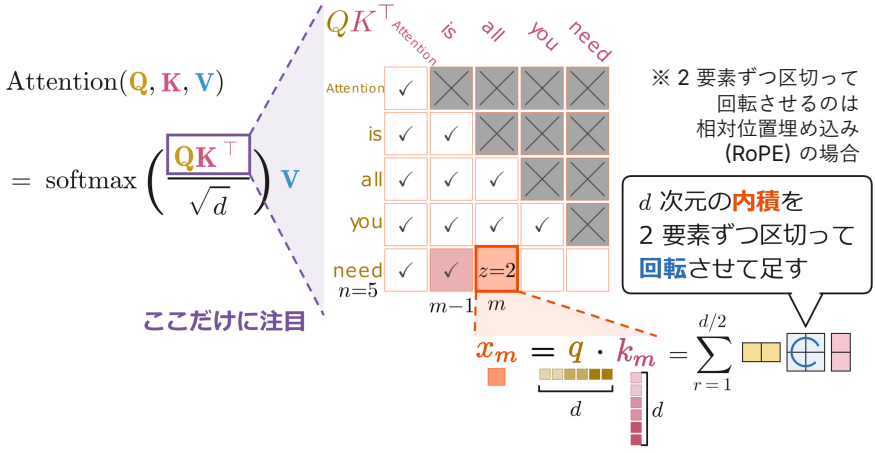
背景: ローカル LLM 推論のボトルネックはどこか



	Attention	FFN/MoE	律速
① **Prefill**(KV キャッシュ充填等)	$O(N^2)$	$O(N)$	**計算**
② **Decode**(1 トークン生成)	$O(N)$	N 非依存	**メモリ帯域**

- **課題**: コンテキスト長 N[token] を伸ばすほど
  ⇒ **相対的に Attention が重くなる**
- **Decode** は毎ステップ **KV キャッシュを読み直す**
  速度は**メモリから読む量**で決まる

前提：Attention は行列積と softmax — softmax を外すと Linear Attention



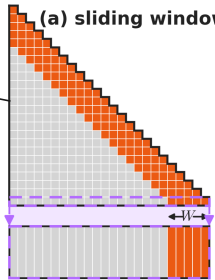
$$\mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top)\,\mathbf{V} \;\approx\; \mathbf{Q}(\underbrace{\mathbf{K}^\top\mathbf{V}}_{\text{固定サイズ } d\times d(\text{例 } d=128)})$$

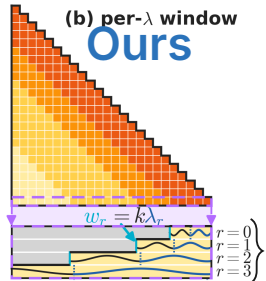
Linear Attention・SSM・GDN

最近のモデルは層の一部 (例：4 層のうち 3 層) がこちらに置き換わり、**KV キャッシュが増えにくい**

ATFlash Kernel [2]: RoPE の波長に比例した距離フィルタ — トークンは捨てない

スライディング ウィンドウ
時系列で古い単語が
トークンごと消える





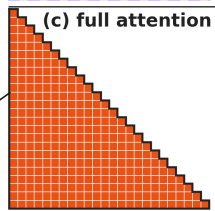
本手法: 切る距離が
周波数ごとに違う
⇒ **低周波ほど遠くまで残る**

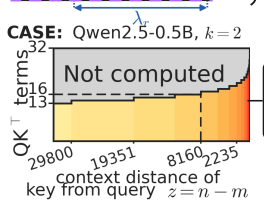
$$w_r = \min(k\,\lambda_r, N)$$

学習不要のパラメタ k

色の濃さ = **残る内積の項数**

最終クエリ行を
周波数ごとに**分解**して
可視化したもの





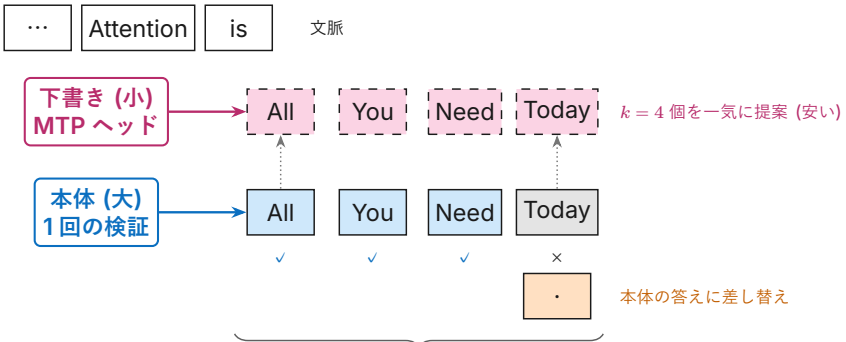
実データ: 遠方でも
32 本中 **13 本は残る**

三角形 = $\mathbf{QK}^\top$
行 = $\mathbf{q}$(クエリ)
列 = $\mathbf{k}$(キー)

[2] S. Hayashi et al., “ATFlash: Per-RoPE-Wavelength Attention Windows for Compute/Memory-Efficient LLM Inference”, arxiv.org/abs/2608.02947

MTP(Multi Token Prediction): 投機デコードで生成を速くする

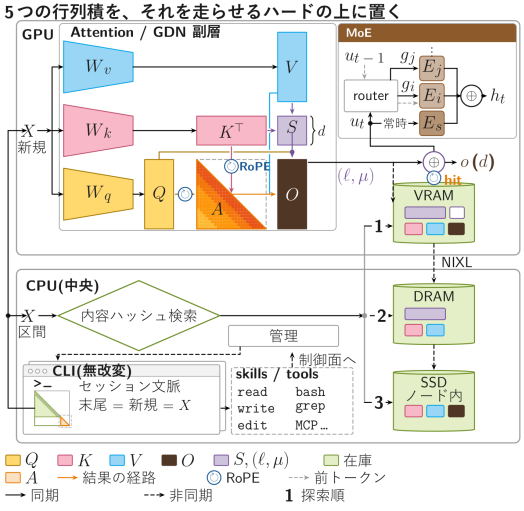
MTP ヘッドは**モデルに同梱**されて配られることが多い
別の下書きモデルを用意しなくても**投機デコード**が使える



当たった分だけ進む：3 + 1 = **4 トークン**を、重みを 1 回読むだけで生成
外れは本体の答えに差し替えるので品質は変わらない

詳細は Zenn の記事 (右): `zenn.dev/katalab/articles/386df3380c2888`

ATFlash Engine(開発中): マルチエージェントの個人利用に特化した stateful なエンジン



前提：1 つのジョブ割当の中で
推論と計算が同じ GPU を使う

課題 1: LLM サーバは起動が遅く、
止めると立ち上げ直しになる。
計算ジョブもすぐには流れ始めない
→ **動的な割当**: サーバは常駐したまま、
推論の要求が途切れた間だけ
VRAM を計算へ渡し、要求が来たら取り戻す

課題 2: 既存のエンジンは
prefix が完全一致する所しか再計算を省けない
→ **状態の保持**：KV をエージェント間で共有
GDN・SSM の状態も VRAM・DRAM・SSD へ

想定：VibeCodeHPC のように
1 人が多数のエージェントを回す使い方

まとめ

1. **ハーネス = LLM の性能を存分に引き出す仕組み。**ソフトも設定も運用も画面操作も含む。
   その外側が **リソースエンジニアリング**
2. **作っているもの**
   (i) 手元でもスパコンでも動く**メタハーネス + カスタムハーネス**
   — VibeCodeHPC(適宜アップデート)
   (ii) スパコン内部でローカル LLM だけで完結するときの**困りごとを克服する**
   — ATFlash Engine(研究中)
3. **研究自動化**：HPC-AutoResearch と ARI
   — アイデア出し・先行研究の調査から、環境構築・ジョブ投入・評価・執筆まで通して回す