

生成 AI によるスーパーコンピュータのプログラム開発 — HPC-GENIE プロジェクトの紹介

棕木大地（助教）

林俊一郎（M1），森田光貴（M1），磯部晃輝（B4），樹神宇徳（B4），阪口修吾（B4），
三笠諒（B4），星野哲也（准教授），片桐孝洋（教授）

所属はすべて名古屋大学

2026 年 3 月 16 日

大学等におけるオンライン教育とデジタル変革に関するサイバーシンポジウム
「教育機関 DX シンポ」（オンライン）

スーパーコンピュータ (スパコン)

多数のコンピュータを高速なネットワークで接続したクラスター型計算機が主流

近年は **GPU** 搭載システムが主流

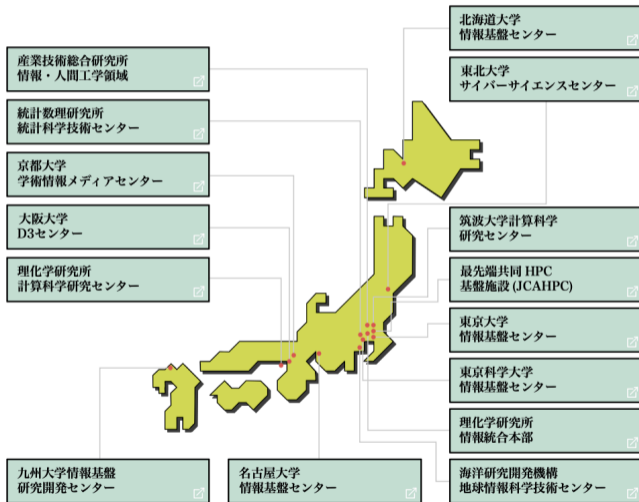
一つの計算を**並列処理**で超高速に計算

用途：科学技術計算，シミュレーション，**AI の学習・推論**，etc.

スパコンに関連する技術分野 – **高性能計算** (High Performance Computing, **HPC**)



革新的ハイパフォーマンス・コンピューティング・インフラ (HPCI)



※図の出典：https://www.hpci-office.jp/about_hpci/what_is_hpci

AI 最前線

博士レベルの知識を持つ AI – GPT-5.4, Claude Opus-4.6, Gemini 3.1 Pro...
対話型から **Agentic AI** へ – 自律的に計画・ツールを利用し目標達成

AI コーディング

1. AI 支援型エディタ
 - ┆ Copilot (拡張型), Cursor (IDE 統合)
2. **CLI エージェント** – Claude Code, Codex, Gemini CLI
 - ┆ コード開発のためのツールを使いこなし実装・実行・検証を自律的に行う
 - ┆ 2025/5 の Claude Code with Claude 4 で火がつく? { まだ 1 年
3. **Vibe Coding** – AI との自然な会話によるコーディングスタイル
4. **マルチエージェントシステム**
 - ┆ 複数の AI エージェントを連携させた複雑なタスク処理
 - ┆ Claude Code: Subagents (2025/7) ! Agent Teams (2026/2)

AI コーディングはもはや常識

スパコンプログラミング

C/C++, **Fortran**, Julia

OpenMP (スレッド並列), CUDA (GPU), OpenACC (GPU), MPI (分散並列計算のための通信ライブラリ), etc.

コード開発における課題

性能 (=速度) が重要

多様なアーキテクチャ (CPU, GPU) ・分散並列計算機への性能最適化
専門的な数値計算アルゴリズム, 計算誤差への対処

コーディング AI 活用への期待

コードリファクタリング – **Fortran** で書かれた数十万行のコードが存在

- ↓ スパゲッティ, 開発者・管理者がいない, バグがありコードでサイエンス

富岳 NEXT (2030 年予定) – 日本の旗艦スパコン初の GPU 採用, コードの **GPU 化** が課題

不足する HPC の高度人材・専門知識・技術の補完に強い期待

HPC-GENIE プロジェクト ^a

High-Performance Computing with **GEN**erative Neural Intelligence for **EX**ecution

HPC コード開発への生成 AI 応用

名古屋大学 情報基盤センター (2025 年 4 月～)

重点的な研究テーマ

エージェントシステム

- 1 複数 LLM が連携動作するマルチエージェント
- 1 長時間の自律動作

ローカル LLM

- 1 商用サービス依存脱却 { ロックイン防止, コスト削減
- 1 セキュリティ { コードやデータの流出防止
- 1 オープン・国産技術開発

^ahttps://www.hpc.itc.nagoya-u.ac.jp/menu/hpc_genie.html



GPT-4.1/o4-mini による BLAS ルーチンコードの生成¹⁾

BLAS (Basic Linear Algebra Subprograms) – 基本線形代数演算のルーチン群
多くのルーチン名だけから動作コードを生成 (zero-shot で 10 回試行)
ネット公開のコードとは異なる構造 (コード最小化の傾向) – LLM が仕様を理解

約 1 年前の汎用 LLM でも科学技術計算向けライブラリの仕様を理解

Claude Code による Intel GPU 向け SYCL 言語コード生成²⁾

CPU 向け BLAS コードの SYCL 言語による Intel GPU 化
主流の NVIDIA GPU (CUDA) 以外もいける

AI によるコード移植 – ハードウェアのベンダーロックイン回避への期待

¹⁾D. Mukunoki et al., Performance Evaluation of General Purpose Large Language Models for Basic Linear Algebra Subprograms Code Generation, arXiv:2507.04697, Jul. 2025.

²⁾K. Morita et al., Evaluation of the Capability of Coding AI in Generating SYCL-Based Numerical Computation Codes for Intel GPUs, SCA/HPCAsia2026, poster session, Jan. 2026.

AI による Fortran コードの GPU 化 (1)

Claude Code による GeoFEM/Cube の GPU 化 ^a

Fortran 90 の FEM コード (MPI+OpenMP) を
Claude Code (Opus 4.1) で GPU 化

対象 GPU : NVIDIA GH200 (東大・Miyabi-G)

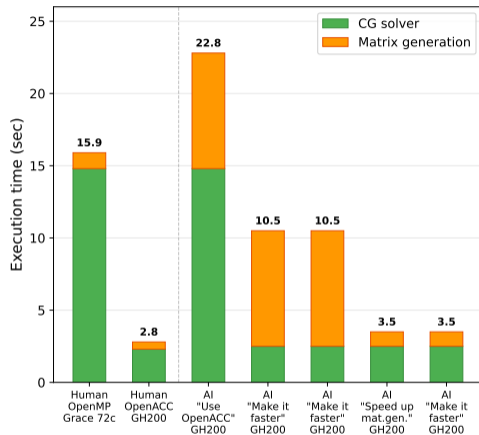
主な構成要素 : 係数行列生成 + CG ソルバー

CG ソルバー (典型的) ! 簡単に成功

係数行列生成 (独自コード) ! しばしば失敗

典型的なコードは得意, 独自ロジックは苦手

「もっと高速化せよ」は有効



最適化の推移 (縦軸: 実行時間)
左: 元コード, 左2: 人力最適化
左3-右: AI 最適化

^aT. Hoshino et al., Evaluating Claude Code's Coding and Test Automation for GPU Acceleration of a Legacy Fortran Application: A GeoFEM Case Study, LLM4HPCAsia 2026.

AI による Fortran コードの GPU 化 (2)

3 社の CLI エージェントによる ICTCG 法コードの GPU 化³⁾

Fortran 90 の疎行列ソルバーコード (OpenMP, 約 3500 行)

Claude Code (Opus 4.5), Codex (gpt-5.2-codex), Gemini CLI (gemini-2.5-pro) を比較

脱最適化の導入: CPU 固有の最適化を除去

GPU 移植結果 (CPU 比のスピードアップ)

入力コードの状態	Claude	Codex	Gemini
オリジナル (CPU 最適化済み)	2.6	失敗	失敗
Stage-1 (マクロ・コメント除去)	-	失敗	失敗
Stage-2 (OpenMP 除去 ! シリアル化)	-	6.3	失敗

脱最適化により AI の GPU 移植成功率が向上

クリーンなコードを維持 ! **AI がコンパイラとして各ハードウェア向けに最適化**

³⁾ 阪口ら, ICTCG 法の計算プログラムの GPU 移植を対象としたコード生成 AI の比較, 情報処理学会第 88 回全国大会, 2026 年 3 月.

VibeCodeHPC^{ab} (主開発者：林俊一郎 (M1))

Claude Code を用いたスパコン向けコード開発・自動最適化システム

マルチエージェントシステム：複数の AI によるチームワーク (PM, SE, PG, CD ...)

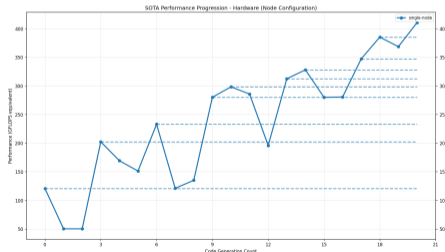
Vibe Coding + 長時間の自律動作を実現

マルチエージェント黎明期の 2025 年夏に開発

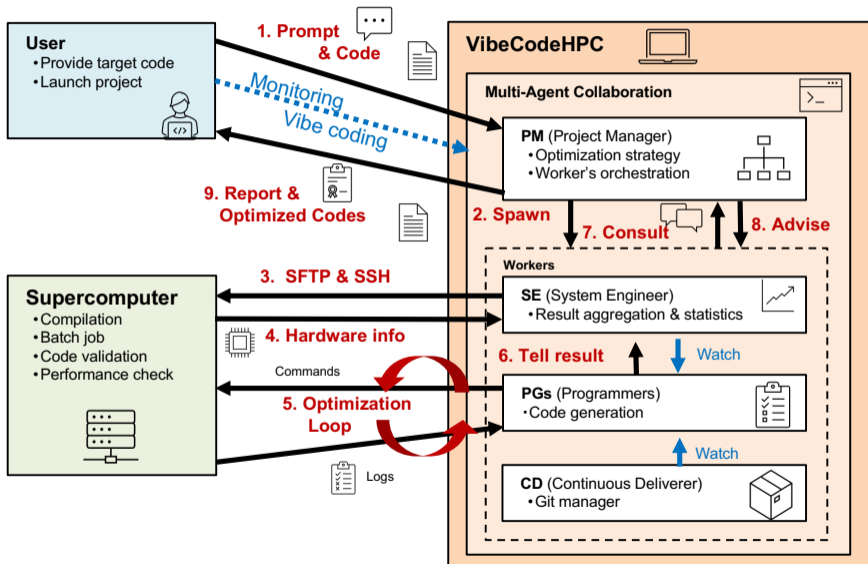
HPC 業界初の
スパコン特化マルチエージェントシステム

^a<https://github.com/Katagiri-Hoshino-Lab/VibeCodeHPC-jp>

^bS. Hayashi et al., VibeCodeHPC: A Multi LLM Agent System for HPC Code Auto-Tuning, iWAPT 2026 (accepted) (arXiv:2510.00031).



VibeCodeHPC – ワークフロー

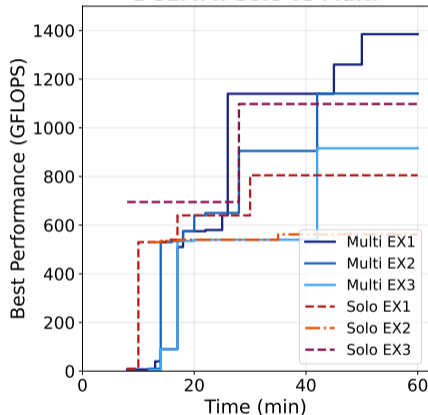


性能 (単位: GFlops/s, n=2048)

技術	モード	実験 1	実験 2	実験 3
OpenMP	マルチ	105	213	190
	ソロ	{	{	{
MPI	マルチ	7	31	{
SIMD	マルチ	{	184	{
CUDA	マルチ	1385	1141	916
	ソロ	805	562	1098
OpenACC	マルチ	250	{	{
MPI+ CUDA	マルチ	{	{	332

名古屋大学 スーパーコンピュータ「不老」 Type II
Xeon Gold 6230 (20c) 2+Tesla V100 4

DGEMM: Solo vs Multi



マルチエージェントはより多くの戦略を探索し、より高い性能を達成

コンテキストウインドウ

LLM の短期記憶

いっぱいになると記憶喪失

CLI エージェントの自動圧縮機能は不十分

マルチエージェントの利点

コンテキストウインドウの総量が増加

自分の仕事に集中しコンテキスト汚染を防ぐ

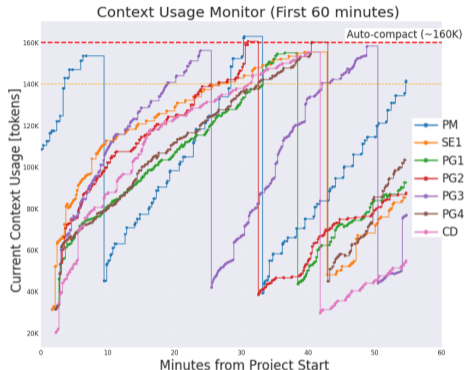
VibeCodeHPC の工夫

各エージェントのコンテキスト消費追跡

– あふれる前に記憶回復指示

エージェント同士の相互監視でミス防止

コンテキスト消費の抑制・記憶整理が課題



VibeCodeHPC における各エージェントのコンテキスト消費追跡

ローカル PC だけで動作するコード最適化システム

なぜローカル LLM か？

商用サービス依存脱却 – ロックイン防止, コスト削減

セキュリティ – コードやデータの流出防止

オープン・国産技術開発

コンシューマ GPU 上の LLM が 7 ヶ月前のトップ商用モデルに匹敵 (EpochAI)

ローカル PC だけで動作するコード最適化システム⁴⁾

OpenAI **gpt-oss-120b** – 128GB メモリで動作 (Ryzen AI Max+/MacBook Pro)

反復最適化 (20 回) – 実装と性能をフィードバックし LLM が次のプロンプトを生成

1. PM (プロジェクトマネージャ) LLM が 5 人の PG (プログラマ) 向けに最適化戦略を生成
2. PG が 5 本のコードを生成
3. 最良コードを選択, PM が最適化効果を分析, 1 に戻る

⁴⁾ 椋木ら, gpt-oss-120b を用いたコード自動最適化マルチエージェントシステムの試作, 第 255 回 ARC・第 202 回 HPC 合同研究発表会, 2025 年 12 月.

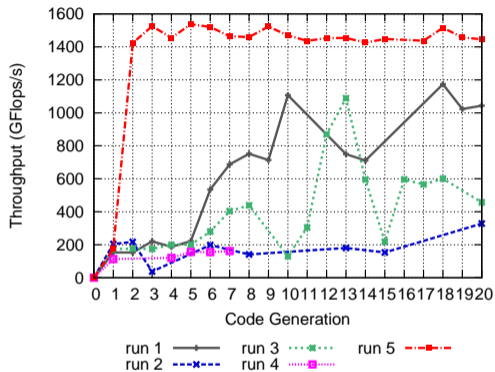
C 言語行列積コードの最適化 (CPU)

入力コード (何も最適化されていない)

```
1 void exp1_opt(int N,  
2             double *A,  
3             double *B,  
4             double *C) {  
5     for (int i = 0; i < N; i++) {  
6         for (int j = 0; j < N; j++) {  
7             double sum = 0.0;  
8             for (int k = 0; k < N; k++) {  
9                 sum += A[i*N+k] * B[k*N+j];  
10            }  
11            C[i*N+j] = sum;  
12        }  
13    }  
14 }
```

実験条件: Intel Xeon Gold 6230 (20c) 2, 正
方形行列 $n = 1024$, 倍精度

Performance Progress - DGEMM



コード性能の推移 (5 回実験)
(縦軸: 性能, 横軸: 反復)

最速のコード (赤線): Intel ライブラリを呼び出し ! **LLM がコードを行列積と認識**
最速の自家製コード (黒線): 1173.42 GFlops/s ! **Intel ライブラリの 76%の性能**

最速の生成コード (行列積)

```
1 static inline void pack_B_panel(const double *B_src, double *B_pack, int N, int kp, int jp, int k_max,
2     int blk_width) {
3     for (int k = kp; k < k_max; ++k) {
4         const double *src = &B_src[k * N + jp];
5         double *dst = &B_pack[(k - kp) * blk_width];
6         int j = 0;
7         __m512_prefetch((const char *)src, _MM_HINT_T0);
8         for (; j < blk_width; j += 8) {
9             __m512d v = __m512_loadu_pd(&src[j]);
10            __m512_storeu_pd(&dst[j], v);
11        }
12        for (; j < blk_width; ++j) { dst[j] = src[j]; }
13    }
14    __m512_sfence();
15}
16 void expl_opt(int N, double *restrict A, double *restrict B, double *restrict C) {
17     const int BLOCK = 64;
18     omp_set_max_threads(omp_get_max_threads());
19     #pragma omp parallel for collapse(2) schedule(static)
20     for (int ii = 0; ii < N; ii += BLOCK) {
21         for (int jj = 0; jj < N; jj += BLOCK) {
22             int i_max = (ii + BLOCK > N) ? N : ii + BLOCK;
23             int j_max = (jj + BLOCK > N) ? N : jj + BLOCK;
24             int blk_width = j_max - jj;
25             int vec_width = blk_width & -31;
26             double C_local[BLOCK * BLOCK] __attribute__((aligned(64)));
27             double *C_local_aligned = (double *)__builtin_assume_aligned(C_local, 64);
28             for (int i = 0; i < BLOCK * BLOCK; ++i) C_local_aligned[i] = 0.0;
29             for (int kk = 0; kk < N; kk += BLOCK) {
30                 int k_max = (kk + BLOCK > N) ? N : kk + BLOCK;
31                 double B_pack_local[BLOCK] __attribute__((aligned(64)));
32                 pack_B_panel(B, B_pack_local, N, kk, jj, k_max, blk_width);
33                 for (int i = ii; i + 3 < i_max; i += 4) {
34                     const double *a_row0 = &A[(i + 0) * N + kk];
35                     const double *a_row1 = &A[(i + 1) * N + kk];
36                     const double *a_row2 = &A[(i + 2) * N + kk];
37                     const double *a_row3 = &A[(i + 3) * N + kk];
38                     double *c_loc0 = &C_local_aligned[(i - ii) * BLOCK];
39                     double *c_loc1 = &C_local_aligned[(i - ii + 1) * BLOCK];
40                     double *c_loc2 = &C_local_aligned[(i - ii + 2) * BLOCK];
41                     double *c_loc3 = &C_local_aligned[(i - ii + 3) * BLOCK];
42                     for (int j = 0; j < vec_width; j += 16) {
43                         __m512d c00 = __m512_loadu_pd(&c_loc0[j + 0]);
44                         __m512d c01 = __m512_loadu_pd(&c_loc0[j + 8]);
45                         __m512d c10 = __m512_loadu_pd(&c_loc1[j + 0]);
46                         __m512d c11 = __m512_loadu_pd(&c_loc1[j + 8]);
47                         __m512d c20 = __m512_loadu_pd(&c_loc2[j + 0]);
48                         __m512d c21 = __m512_loadu_pd(&c_loc2[j + 8]);
49                         __m512d c30 = __m512_loadu_pd(&c_loc3[j + 0]);
50                         __m512d c31 = __m512_loadu_pd(&c_loc3[j + 8]);
51                     }
52                     for (int k = kk; k < k_max; ++k) {
53                         const int k_off = k - kk;
54                         __m512d b0 = __m512_load_pd(&B_pack_local[k_off * blk_width + j + 0]);
55                         __m512d b1 = __m512_load_pd(&B_pack_local[k_off * blk_width + j + 8]);
56                         __m512d a0 = __m512_set1_pd(a_row0[k_off]);
57                         __m512d a1 = __m512_set1_pd(a_row1[k_off]);
58                         __m512d a2 = __m512_set1_pd(a_row2[k_off]);
59                         __m512d a3 = __m512_set1_pd(a_row3[k_off]);
60                         c00 = __m512_fmadd_pd(a0, b0, c00); c01 = __m512_fmadd_pd(a0, b1, c01);
61                         c10 = __m512_fmadd_pd(a1, b0, c10); c11 = __m512_fmadd_pd(a1, b1, c11);
62
```

```
63     c20 = __m512_fmadd_pd(a2, b0, c20); c21 = __m512_fmadd_pd(a2, b1, c21);
64     c30 = __m512_fmadd_pd(a3, b0, c30); c31 = __m512_fmadd_pd(a3, b1, c31);
65     }
66     }
67     }
68     }
69     }
70     for (int j = vec_width; j < blk_width; ++j) {
71         double s0 = c_loc0[j]; double s1 = c_loc1[j];
72         double s2 = c_loc2[j]; double s3 = c_loc3[j];
73         for (int k = kk; k < k_max; ++k) {
74             const int k_off = k - kk;
75             double b = B_pack_local[k_off * blk_width + j];
76             s0 += a_row0[k_off] * b; s1 += a_row1[k_off] * b;
77             s2 += a_row2[k_off] * b; s3 += a_row3[k_off] * b;
78         }
79         c_loc0[j] = s0; c_loc1[j] = s1; c_loc2[j] = s2; c_loc3[j] = s3;
80     }
81 }
82 for (int i = i_max - ((i_max - ii) % 4); i < i_max; ++i) {
83     const double *a_ptr = &A[i * N + kk];
84     double *c_loc = &C_local_aligned[(i - ii) * BLOCK];
85     for (int j = 0; j < vec_width; j += 8) {
86         __m512d c_vec = __m512_loadu_pd(&c_loc[j]);
87         for (int k = kk; k < k_max; ++k) {
88             const int k_off = k - kk;
89             __m512d b_vec = __m512_loadu_pd(&B_pack_local[k_off * blk_width + j]);
90             __m512d a_vec = __m512_set1_pd(a_ptr[k_off]);
91             c_vec = __m512_fmadd_pd(a_vec, b_vec, c_vec);
92         }
93         __m512_storeu_pd(&c_loc[j], c_vec);
94     }
95     for (int j = vec_width; j < blk_width; ++j) {
96         double sum = c_loc[j];
97         for (int k = kk; k < k_max; ++k) {
98             const int k_off = k - kk;
99             sum += a_ptr[k_off] * B_pack_local[k_off * blk_width + j];
100        }
101        c_loc[j] = sum;
102    }
103 }
104 }
105 for (int i = ii; i < i_max; ++i) {
106     double *c_ptr = &C[i * N + jj];
107     double *c_loc = &C_local_aligned[(i - ii) * BLOCK];
108     c_ptr = (double *)__builtin_assume_aligned(c_ptr, 64);
109     c_loc = (double *)__builtin_assume_aligned(c_loc, 64);
110     int j = 0;
111     for (; j < vec_width; j += 8) {
112         __m512d tmp = __m512_loadu_pd(&c_loc[j]);
113         __m512_storeu_pd(&c_ptr[j], tmp);
114     }
115     for (; j < blk_width; ++j) {
116         c_ptr[j] = c_loc[j];
117     }
118 }
119 }
120 }
```

ローカルPCで動作するLLMでも高度に最適化されたコードを生成できる

コードの性能を考慮したモデル学習^a

既存モデルは実機での性能を考慮した学習がされていない

GRPO (Group Relative Policy Optimization) (Shao et al. 2024) を利用

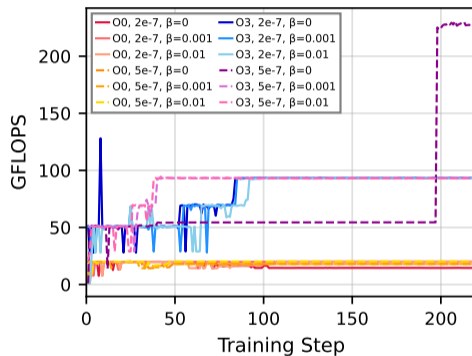
実機でのコード性能を報酬としてフィードバックし学習

Qwen2.5 Coder 14B Instruct をファインチューニング

CPU の行列積コード生成能力強化を実証

対象システムに特化した
性能最適化 LLM を構築可能

^aR. Mikasa et al., Improving HPC Code Generation Capability of LLMs via Online Reinforcement Learning with Real-Machine Benchmark Rewards, arXiv:2602.12049, 2026.



学習の推移

(縦軸：性能, 横軸：学習ステップ)

The AI Scientist^a (Sakana AI 2025)

研究全体を自動化:

アイデア / 実装 / 実験 / 評価 / 論文

Python を用いた AI 研究に特化

HPC 研究への適用 – HPC-AutoResearch^{bc}

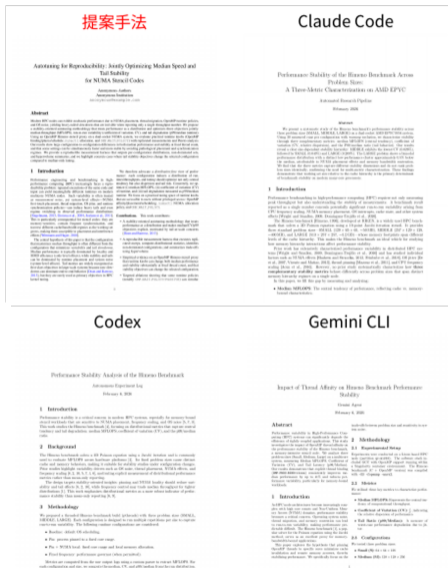
フェーズ分離実行モデルによる HPC 環境での試行
錯誤をともなう環境構築・実験対応

構造化記憶機構による研究フェーズ全体の知識管理
商用 AI エージェントを上回る論文生成を実証

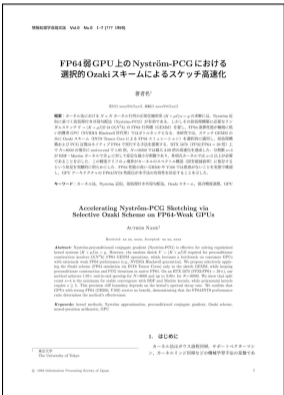
^aM.J. Fontaine et al, The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery, arXiv2412.05210, 2025

^b樹神ら, HPC-AI Scientist: LLM を活用した HPC 向け研究フレームワーク, 第 203 回 HPC・第 17 回 QS 合同研究発表会, 2026 年 3 月.

^c<https://github.com/kotama7/HPC-AutoResearch>



簡単なテーマなら「着想！ 関連研究調査！ 手法提案！ コード実装！ 実験評価！ 分析！ 考察！ 論文執筆！ スライド作成！ 原稿作成」をある程度自動化

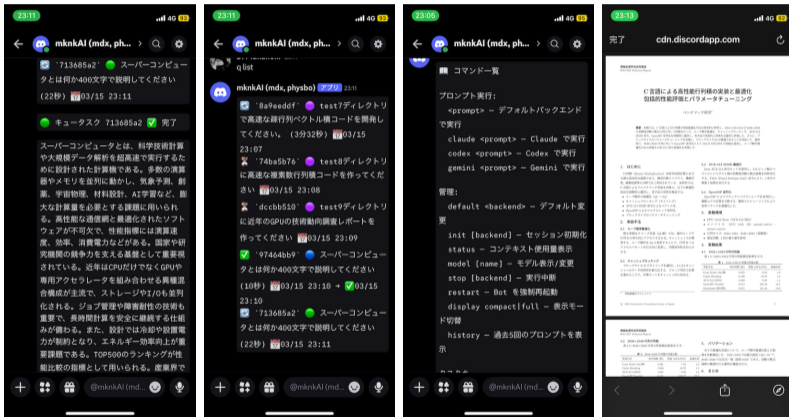


このテーマに関して何も知らない学生が5時間で作った論文！ 学部卒論合格レベル？

Aと対話しながら研究する「Vibe Research」の時代へ

Discord Bot による CLI エージェント操作

Discord bot によりサーバで動いてる Claude Code/Codex/Gemini CLI をチャットで操作



スマホからチャットでスパコンを操作し Vibe Coding・Vibe Research!

HPC-GENIE プロジェクト（名古屋大学）

HPC コード開発における生成 AI の活用に関する研究
エージェント開発とローカル LLM に注力

HPC コード開発における生成 AI への期待

AI がコンパイラになる未来 – クリーンなコードだけ保持すればよい

「富岳 NEXT」に向けた Fortran コードの GPU 化

HPC だけでなく物理学や数学を理解 – 「博士号レベル」の知識はあらゆる分野に．計算内容やアルゴリズムを理解した開発・デバッグも可能

課題

責任の所在，無理解，教育・人材育成，著作権・ライセンス，商用サービス依存...

生成コードの検証 – 本質的には人間と同じ，むしろ AI 支援に期待

超高速の技術発展にどうついていくか（道具として・研究として）

「AI のためのスパコン」だけでなく「スパコンのための AI」の研究へ