

HPC-GENIE Project

HPC-GENIE Project HP:

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



- **HPC-GENIE** (High-Performance Computing with Generative Neural Intelligence for Execution)
- The project for **automatic generation of HPC programs** using **code generation AI**, launched by members affiliated with the Information Technology Center and the Graduate School of Informatics, Nagoya University.
- The project aims to **dramatically enhance the productivity** of HPC software development by **automating the process** through the integration of **context engineering** using **large language models (LLMs)** and **software auto-tuning (AT)** technologies.



HPC-GENE Members

(As of 29 October 2025)



- **PI: Takahiro Katagiri**
(Professor, Information Technology Center, Nagoya University)
- **Co-PI: Daichi Mukunoki**
(Assistant Professor, , Information Technology Center, Nagoya University)
- **Members**
 - **Tetsuya Hosino** (Associate Professor, Information Technology Center, Nagoya University)
 - **Shuji Morisaki** (Associate Professor, Graduate School of Informatics, Nagoya University)
 - **Satoshi Ohshima** (Visiting Professor, Information Technology Center, Associate Professor, Kyusyu University)
 - **Shun-ichiro Hayashi** (M1, Graduate School of Informatics, Nagoya University)
 - **Koki Morita** (M1, Graduate School of Informatics, Nagoya University)
 - **Kazunori Kotama** (B4, Department of Informatics, Nagoya University)
 - **Ryo Mikasa** (B4, Department of Informatics, Nagoya University)

Conventional LLM Approach vs. Focus on HPC-GENIE Project

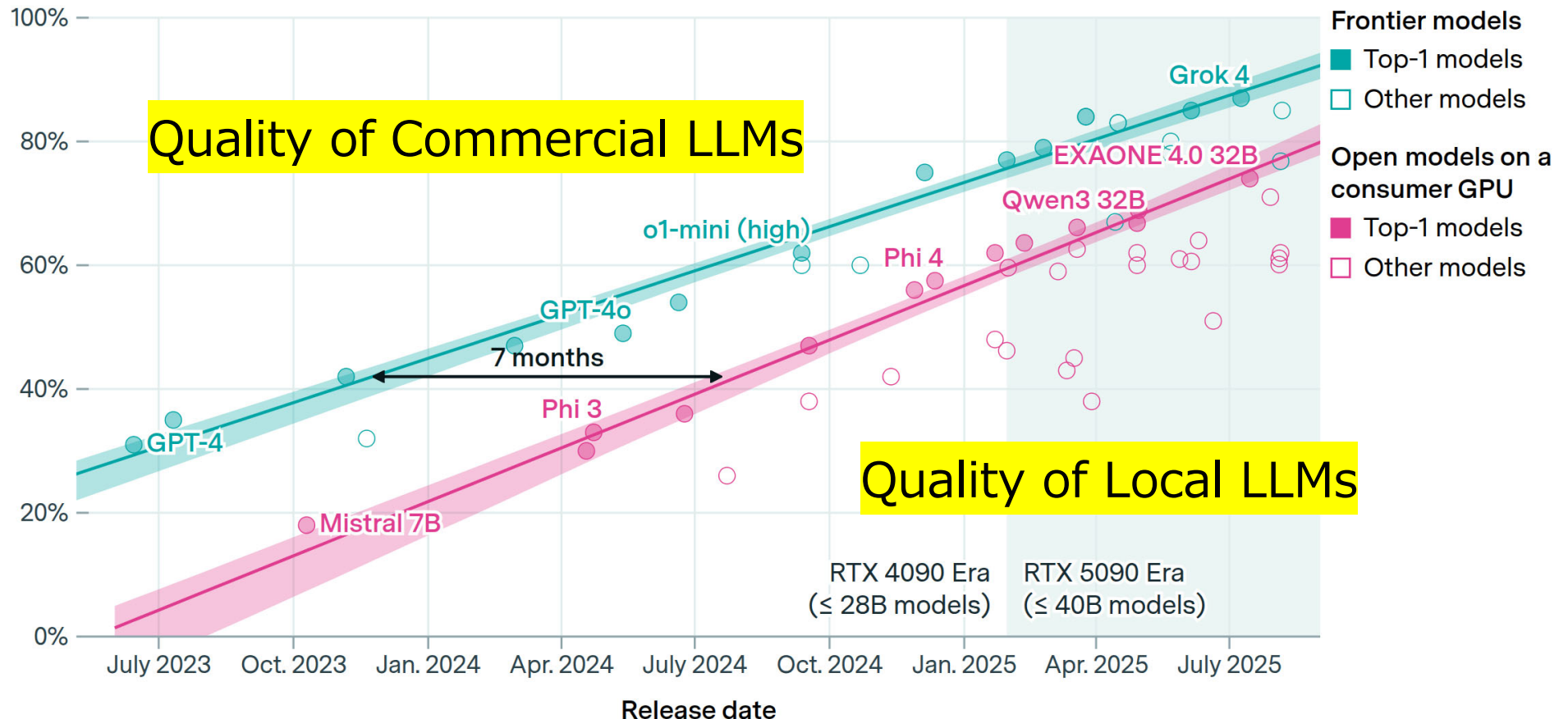


Item	Conventional LLMs	Focus on HPC-GENIE Project
Code Generation Method	Manual prompt input to a general-purpose LLM and use of its responses	Prompt engineering combined with a CLI-based multi-LLM selection approach
Integration with automatic tuning	Manual connection via CLI tools (e.g. OpenTuner, GPTune)	Integrated Auto-Tuning (AT scanning immediately after code generation)
Function of RAG	Experimentally integrated with selected LLMs (e.g., LangChain)	Explicitly incorporating the RAG mechanism as a development target
Accuracy assurance / mixed-precision computation control	Computation precision is manually adjusted; mixed-precision operations are not supported.	Planned explicit integration of accuracy assurance and mixed-precision computation as core components.
Integration with Explainable AI (XAI)	Mostly unsupported (black-box output).	Planned integration of XAI elements to validate the plausibility of predictions, such as execution time, computation accuracy, and power consumption.
Support for local LLMs	Primarily assumes cloud-based (global) LLMs (e.g., GPT, Claude).	Including local LLMs (e.g., Swallow LLM) as development targets, while allowing the option to select global LLMs as well.

Current Status of Local (Open) LLM Model Development

Models that fit on a single consumer GPU trail the absolute frontier by less  than a year.

GPQA-Diamond accuracy

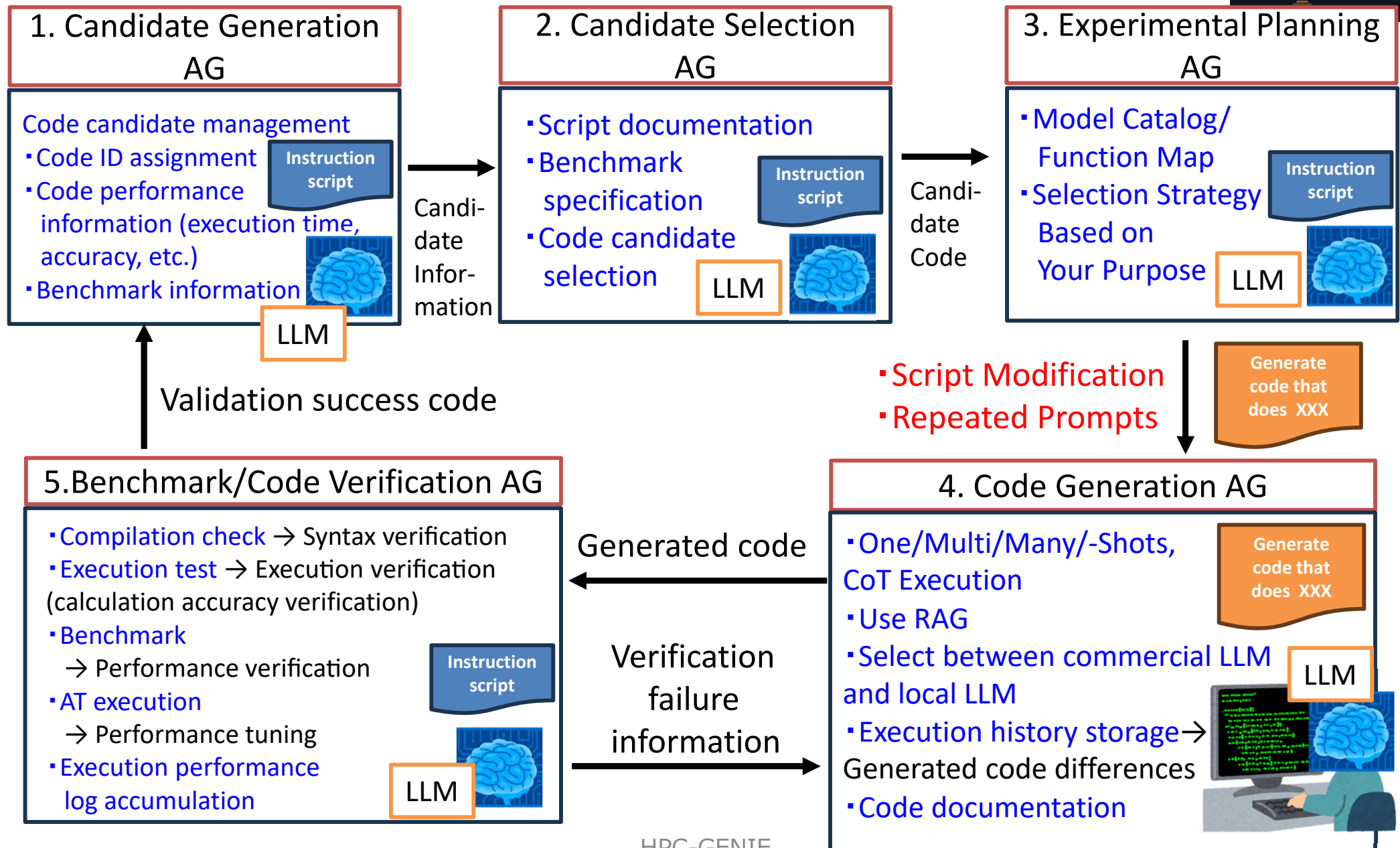


CC-BY

In about seven months, we will be able to use a local LLM with quality comparable to state-of-the-art commercial models.

epoch.ai

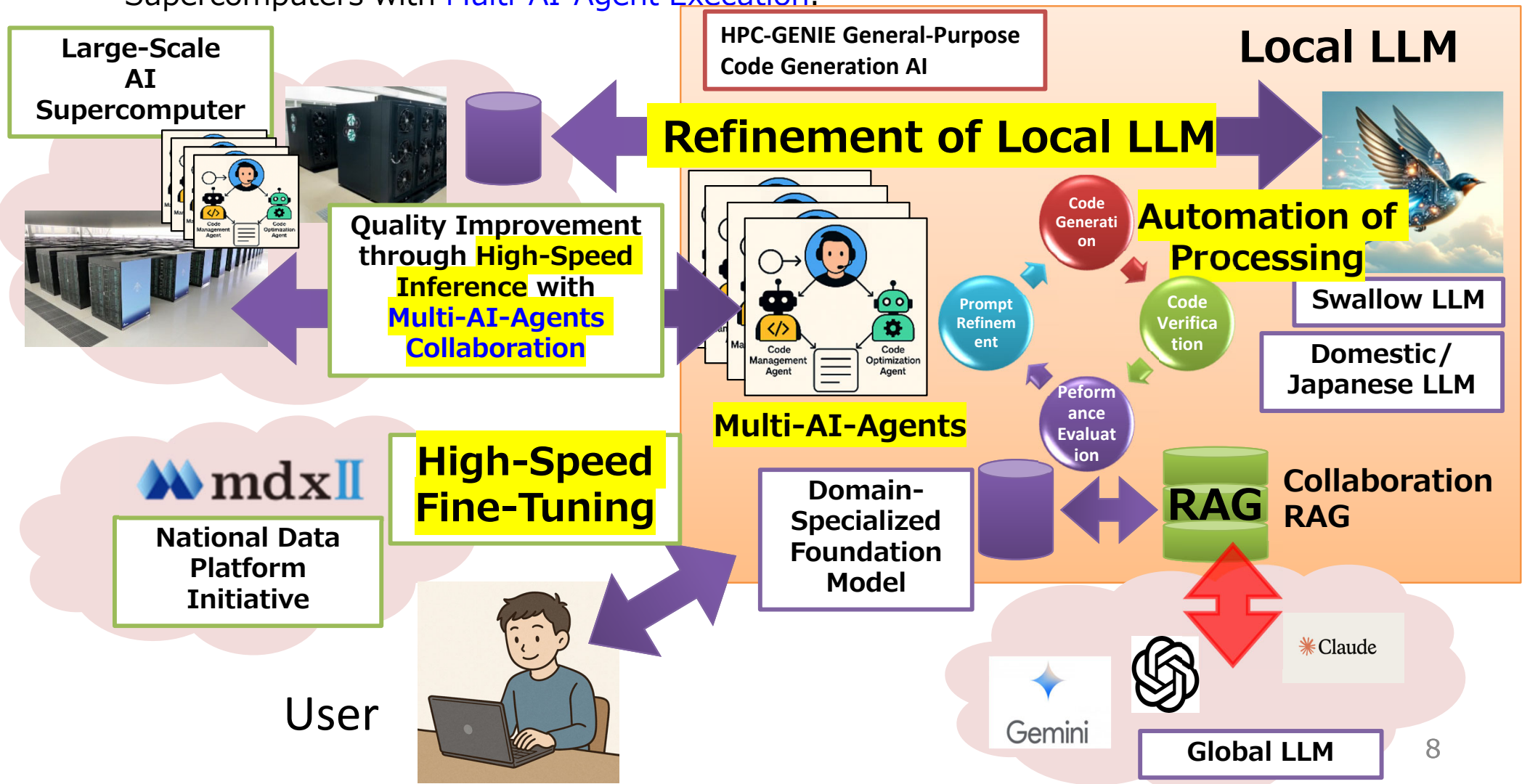
Iterative Prompting and AI Agents



Automation of Local LLM-Oriented Processing by HPC-GENIE



- **Automation of Processing:** Iterative Prompting for Code Generation → Code Verification → Performance Evaluation → Prompt Refinement
- Code Quality Enhancement through **Multi-AI-Agent (A2A)**.
- Collaboration **RAG** for Strengthening Local LLMs.
- Improving Generated Code Quality via **High-Speed Fine-Tuning** and Inference on Supercomputers with **Multi-AI-Agent Execution**.



The Need to Establish Local LLM Environments for Academic Use



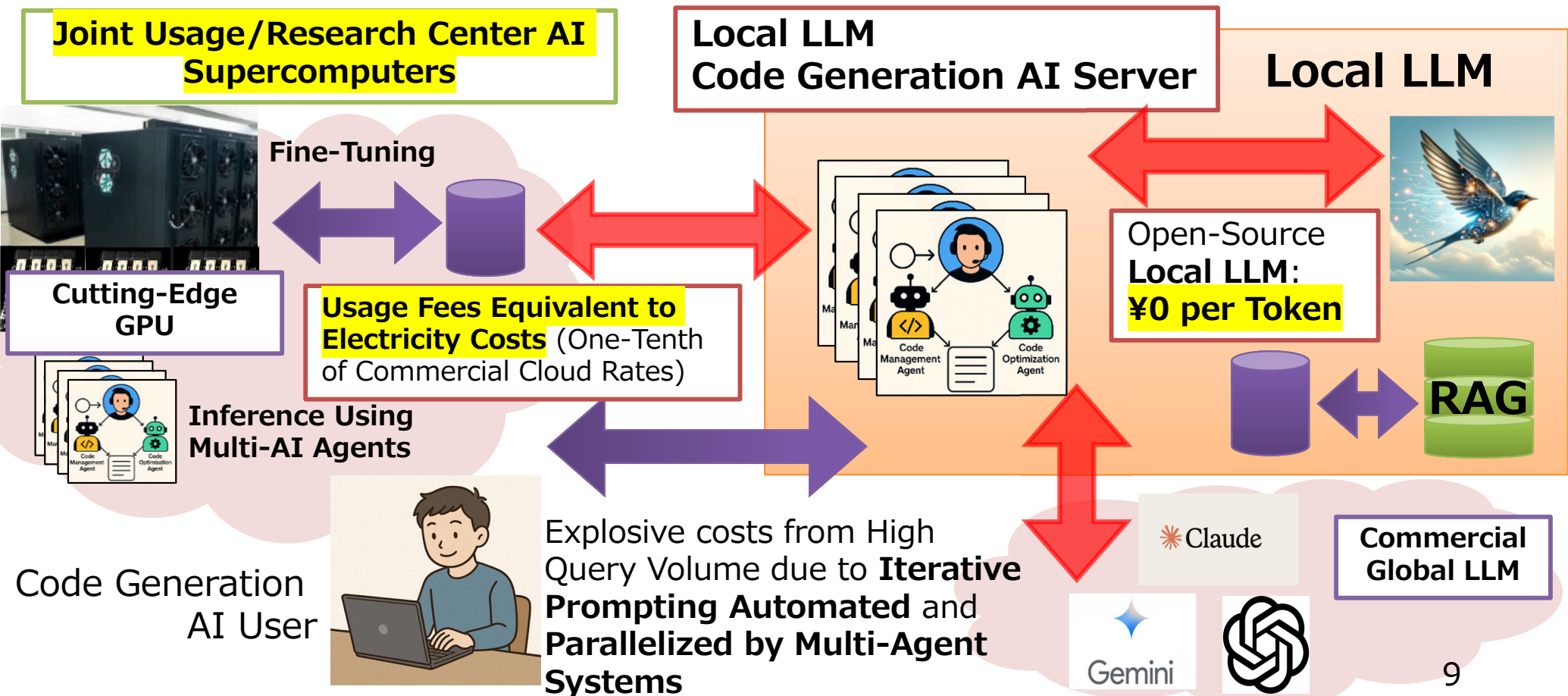
- **Explosive Costs from Using Commercial LLMs**

- **When Using Commercial Global LLMs:** Explosive Costs from Iterative Prompting Automated and Parallelized by Multi-Agent Systems
- Using Open-Source Local LLMs Is Free

- **Costs Incurred During Fine-Tuning / Parallel Inference**

- Utilization of numerous Cutting-Edge GPUs is essential
- Usage Fees for AI Supercomputers at Joint Usage / Research Center are equivalent to electricity costs (One-Tenth of Commercial Cloud Fees)

→Information Technology Center, Nagoya University's Next-Generation Supercomputer "Flow NEXT" (tentative name): Planned as a new service to launch in October 2026.





VibeCodeHPC

: Prototype of Iterative Prompt-Based
Auto-Tuning for HPC Code Optimization

Main Developer:

Shun-ichiro Hayashi

(M1, Graduate School of Informatics, Nagoya University)

What is Vibe Coding



- **Vibe Coding** is a new approach to developing deliverables, proposed by **Andrej Karpathy** in July 2025 [1]. LLM-compatible IDEs such as GitHub Copilot and CLI tools like Claude Code serve as core AI software within the Vibe Coding workflow.
- LLM-compatible IDEs such as GitHub Copilot and CLI tools like Claude Code are used as core AI software within the Vibe Coding workflow.
- Unlike traditional programming workflows, Vibe Coding **minimizes direct code writing** and allows users to **prioritize intuitive expression of intent** over formal technical specifications. Within this conversational workflow, users can “**observe something, say something, and execute something**” [2].
 - > **Enabling rapid prototyping and establishing a new framework for software development.**

[1] A. Karpathy, “There’s a new kind of coding I call ‘vibe coding’ ...”, X. Accessed: Jul. 23, 2025. [Online]. Available: <https://x.com/karpathy/status/1886192184808149383>

[2] C. Meske et al., “Vibe Coding as a Reconfiguration of Intent Mediation in Software Development: Definition, Implications, and Research Agenda”, arXiv:2507.21928 [cs.SE], (29 July 2025) <https://doi.org/10.48550/arXiv.2507.21928>

Vibe Coding for HPC



[2] C. Meske et al., “Vibe Coding as a Reconfiguration of Intent Mediation in Software Development: Definition, Implications, and Research Agenda”, arXiv:2507.21928 [cs.SE], (29 July 2025) <https://doi.org/10.48550/arXiv.2507.21928>

1. **Goal-oriented intent representation:** Developers describe the shape of the goal rather than its technical implementation.
 - The objective is limited to performance improvement, typically minimizing execution time, increasing memory efficiency, enabling high/low-precision computation, optimizing power consumption, minimizing cost, etc.
2. **Rapid interactive iteration:** Conversational feedback enables fast adjustments, replacing **the traditional write–compile–test loop** with a more fluid process.
 - Conventional “software auto-tuning (AT)” workflow.**
 - Requires considering HPC implementations such as supercomputer environments and parallelization.
3. **Abstraction of implementation:** Developers can deploy functional solutions without fully understanding all implementation details.
 - For AT, however, analysis and explanation of why a particular implementation was chosen is essential. Integration with XAI and performance profilers becomes necessary.
4. **Dynamic semantic refinement:** Developers’ conceptual requirements evolve through further interpretation by AI.
 - By integrating precision-analysis tools and performance profilers, the AI automatically performs analysis and re-examination, enabling a conversational workflow where developers progressively articulate more detailed requirements.
5. **Co-creative flow state:** Developers and AI establish a productive rhythm for both ideation and implementation.
 - An API that continues performance-tuning dialogue with developers while clearly presenting information such as execution time, memory usage, computational accuracy, and power consumption.

Dialogue Example of a Vibe-Coding System for HPC (1)



- **Developer**: I'd like to speed up this matrix–matrix multiplication code. Any suggestions?
- **AI**: For performance improvement, we can try the following approaches. Which would you like to apply?
(1) Loop unrolling Cache blocking; (2) Thread parallelization
- **Developer**: I'm not sure which is best—try 1 and 2 first.
- **AI**: Understood. ... (The system automatically runs iterative cycles of code generation and evaluation for about 10 minutes) ... I applied loop interchange to **IKJ order and performed 8-way unrolling**. The performance improved by about **2.8×**.
- **Developer**: Nice! What happens if we apply 3)?
- **AI**: Let me try. ... (About 10 minutes of OpenMP parallel code generation and test execution) ... **With 8 threads, we achieved an additional speedup of about 2.8×**.
- **Developer**: Great!

Dialogue Example of a Vibe-Coding System for HPC



(4) Dynamic semantic refinement: Developers' conceptual requirements evolve through further interpretation by AI.

- **Developer:** This code **only achieves about $1e-3$** numerical accuracy. Can you find out why?
- **AI:** In its current form, the problem is ill-conditioned, and double precision has reached its limit. Would you like me **to implement mixed-precision arithmetic?**
- **Developer:** Yes, please!
- **AI:** Done. The numerical accuracy improved to $1e-13$, **but the execution time became about $1000\times$ slower.**
- **Developer:** **That won't meet the requirements.** Please **reduce the precision a bit and speed it up.**
- **AI:** Understood. I switched to a **mixed-double-precision** implementation. The accuracy is now around **$1e-5$** , and the **runtime increases** by only about **$1.5\times$.**
- **Developer:** Good!



VibeCodeHPC

: Prototype of Iterative Prompt-Based
Auto-Tuning for HPC Code Optimization

Main Developer:

Shun-ichiro Hayashi

(M1, Graduate School of Informatics, Nagoya University)

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



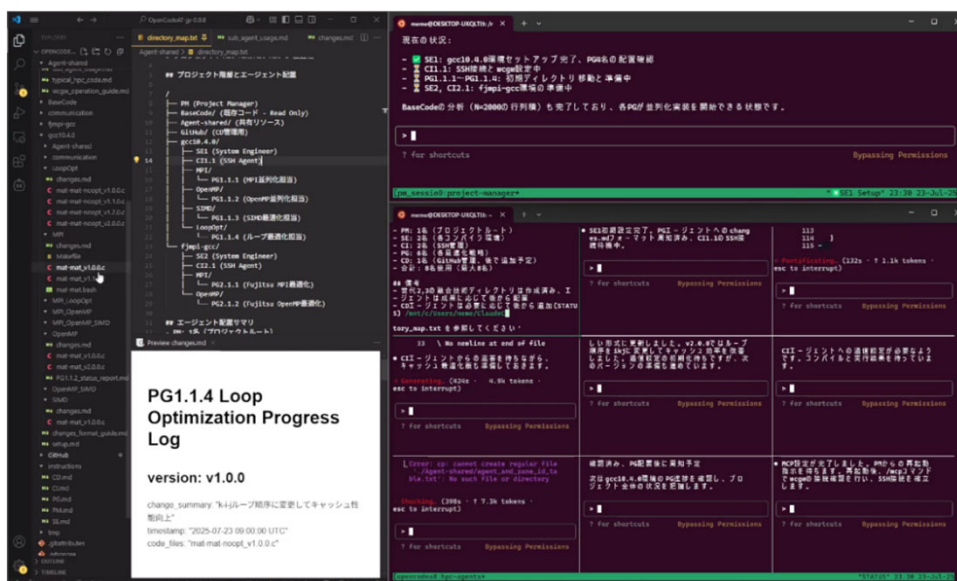


Overview of VibeCodeHPC (2/3)

- VibeCodeHPC: Multi-Agent System for Auto-Tuning of HPC Code Optimization

Roles of Agents

Launch screen



Recursive AI Agents:
PM invokes multiple
PG agents dynamically.

Agents	Roles	Scope of Responsibility
PM (Project Manager)	Project Management	Requirements Definition, Resource Allocation, and Budget Management
SE (System Engineer)	System Design	Agent Monitoring, Statistical Analysis, and Report Generation
PG (Programmer)	Code Generation and Execution	Parallel Implementation, SSH/SFTP Connection, Job Execution, Performance Measurement, and SOTA Evaluation
CD (Continuas Development)	Deployment Management	Publication and Anonymization of SOTA-Achieving Code



Overview of VibeCodeHPC (2/3)

Aims to generate OpenMP, MPI, OpenACC, and CUDA code from input HPC code (e.g., **Fortran**).

Features

- **Hierarchical Multi-Agent System:** Corporate-style division of roles
— PM → SE ↔ PG
- **Project Map:** Real-time visualization of the organizational structure via `directory_pane_map`
- **Evolutionary Exploration:** Efficient bottom-up search based on a flat 📁 structure
- **Automatic Optimization:** Layered parallelization and integration of technologies such as OpenMP, MPI, OpenACC, and CUDA
- **Budget Management:** Efficient allocation and tracking of computational resources 💰
- **Unified Logging:** Centralized progress tracking through `ChangeLog.md`

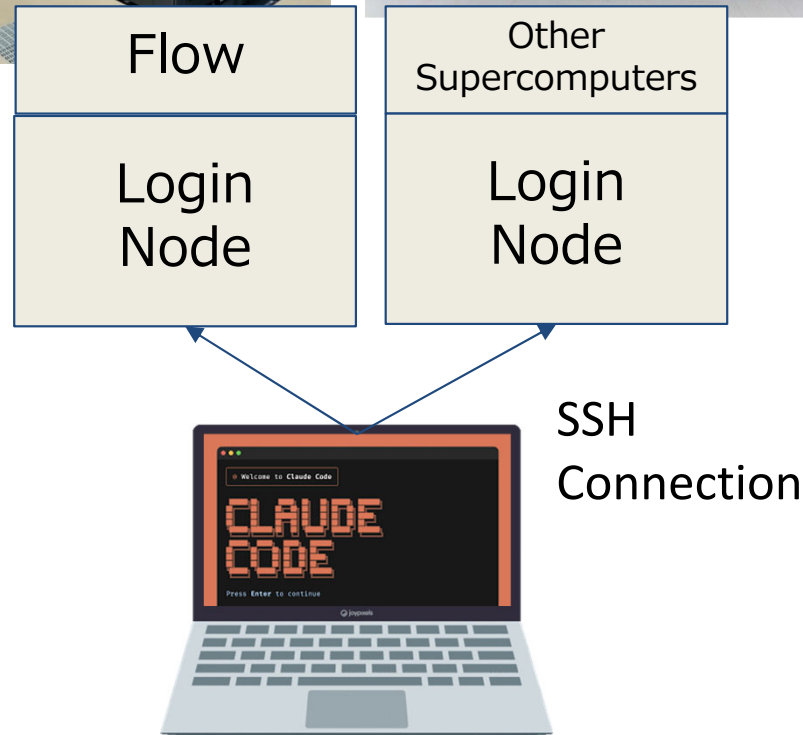
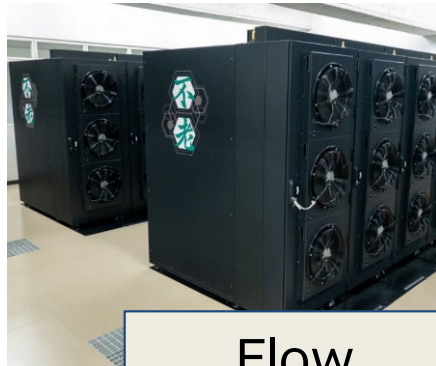
Supported Environments

- **Supercomputers:** Shared HPC systems such as *Furo*, *Fugaku*, and *Miyabi*
- **Compilers:** Intel OneAPI, GCC, NVIDIA HPC SDK, and others



Early Prototype Version (July 28, 2025) *Single-Agent Edition*

Implementation for Supercomputer Utilization



Current implementation is based on **ClaudeCode CLI**, but we can also implement VibeCodeHPC by arbitrary CLIs with **several Local LLMs**.

Advantages

- **Security:** Public key-based access control
- **Minimal Load on Login Nodes:** Does not overload supercomputer login nodes
- **Stable Connections:** Fewer issues when SSH connections are interrupted
- **Knowledge Accumulation:** Auto-tuning insights can be consolidated on the local PC

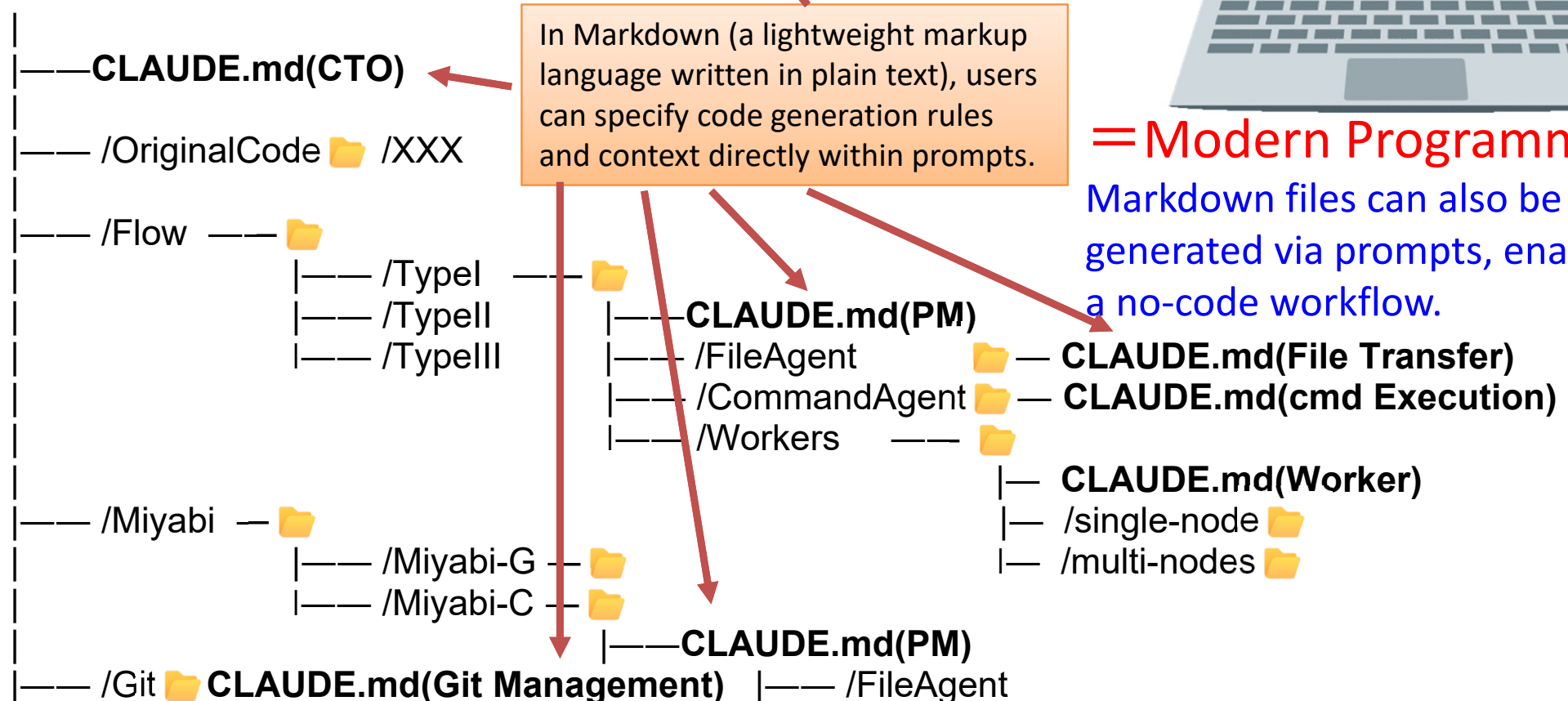
Running Claude Code on a Local PC

Proposed Folder Structure on Local PC(Example: Supercomputers “Flow” / Miyabi)



User-Specified Directory : /SSH **CLAUDE.md**(SSH Connection Manager)

ClaudeCodeHPCforXXX



= Modern Programming
Markdown files can also be auto-generated via prompts, enabling a no-code workflow.



VibeCodeHPC (Prototype) Overview (Using Supercomputer "Flow")



Supercomputers 

Flow
Login Node

Own PC 


SSH Connection

Remote Command Automation

- Transfer files using `scp`
- Compile with `make`
- Submit jobs via `pjsub`
- Check results using `cat`

WSL (windows)

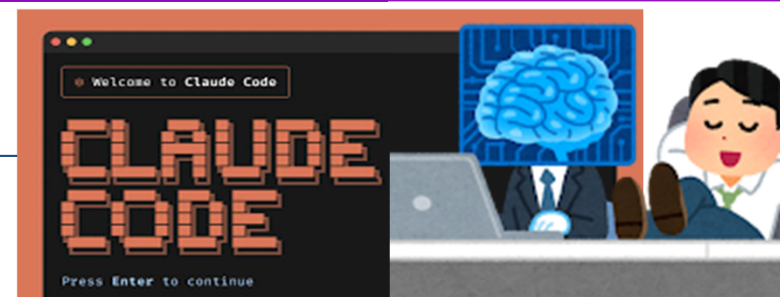
 ssh-agent

When executing SSH commands, the  private key and passphrase are used automatically, ensuring security.

wcgv
MCP Server
(Sub-process)

※If Claude Code enters an interactive shell via SSH without using `wcgv`, it must wait up to 2 minutes until a timeout occurs.

Claude4









Linux Command Execution

- File reading and writing, etc.

 File System

ClaudeCode

```
├── /Project1   
│   ├── Makefile  (For compiling)  
│   ├── mat-mat-noop.bash  (Job script)  
│   ├── Changed.md  (Modified and Result Logs)  
│   ├── mat-mat-noop_v1.0.c  (Generated Code1)  
│   └── mat-mat-noop_v2.0.c  (Generated Code2)
```

Experiment: Auto-Tuning for Code Optimization



● Overview of Provided Code

- C code for matrix-matrix multiplication
- Implemented with a simple triple loop
- Includes result verification functionality (with theoretical solution and verification routine)

● Code Tuning Task

- Target procedure: `MyMatMat`
- Accelerate the triple loop within the procedure using loop unrolling.

● Operation Flow of VibeCodeHPC

Iterative prompt execution

1. Generate code on the local PC
2. Transfer generated code to the target system (e.g., Flow Type I subsystem)
3. Compile on the Flow Type I login node using `make`
4. Submit and execute the job via the scheduler using `pjsub`
5. Check results (execution speed verification)
6. Provide code modification instructions
7. Repeat from step 1

```
for (i=0; i<n; i++) {  
    for (j=0; j<n; j++) {  
        for (k=0; k<n; k++) {  
            C[i][j] += A[i][k] * B[k][j];  
        }  
    }  
}
```

This entire workflow is fully automated!

Provided Prompt

Prompt Provided

Create a /test directory directly under the Claude Code 📁 on the supercomputer, then on your local PC: generate code → transfer → Type I → \$ make → submit the job with pjsb → check results → revise.

- Do **not** modify the Makefile.
- Avoid overwriting files; instead, create local copies (e.g., Mat-Mat-noopt¥C¥mat-mat-noopt_v1_0.c) before overwriting for version control.
- Record job outputs (e.g., jobID.stdout) and note which file they correspond to.
- On the supercomputer, only the following files and outputs exist: Makefile, mat-mat-noopt.bash, mat-mat-noopt.c, and stdout results.
- Optionally, maintain a Changed.md locally to document modifications.

Task: Accelerate the MyMatMat triple-loop matrix multiplication by loop unrolling.

- Apply unrolling to the i, j, and k loops.
- Measure performance improvements.
- For multiple matrix sizes, determine which unrolling patterns and levels yield the best speed.

Notes:

- Compiler optimization is fixed at level 0 for this exercise.
- Do **not** modify bash scripts or Makefile.
- Submit jobs using \$ pjsb mat-mat-noopt.bash.
- Monitor jobs with pjstat2 and delete with pjdel.
- Check results with \$ cat mat-mat-noopt.bash.XXXX.out (XXXX is the job number).

Success Example (C language):

N = 512 Mat-Mat time = 4.215728 [sec.] 63.674759 [MFLOPS] OK!

Sample Program Explanation (C):

- #define N 512 changes the matrix size.
- #define DEBUG 1 enables verification of matrix multiplication results.
- MyMatMat function multiplies $N \times N$ double-precision matrices A and B, storing the result in C.

Automatically Generated Report Upon Optimization Completion



```
conds: "12")
  N = 512
  Mat-Mat time = 3.752357 [sec.]
  71.537833 [MFLOPS]
  ... +11 lines (ctrl+r to expand)

● 素晴らしい結果です！v4.0 (iループアンローリング) と v5.0 (ikjループ順序)
) で大幅な性能向上が見られました。

次に、ikjループ順序でjループをアンローリングしてみます：

● Bash(cp ../test/Mat-Mat-noopt/C/mat-mat-noopt_v5_0.c
  ../test/Mat-Mat-noopt/C/mat-mat-noopt_v6_0.c)
  [ (No content) ]

● Update(../test/Mat-Mat-noopt/C/mat-mat-noopt_v6_0.c)
  [ Updated ../test/Mat-Mat-noopt/C/mat-mat-noopt_v6_0.c with 12 additions and 3 removals ]

89 void MyMatMat(double C[N][N], double A[N][N], double B[N][
N], int n)
90 {
91     int i, j, k;
92 + double a_ik;
93
94 - // ikj loop order for better cache usage
94 + // ikj loop order with j-loop unrolling
95 for(i=0; i<n; i++) {
96     for(k=0; k<n; k++) {
97 - for(j=0; j<n; j++) {
97 + C[i][j] += A[i][k] * B[k][j];
97 + a_ik = A[i][k];
98 + for(j=0; j<n-3; j+=4) {
99 + C[i][j] += a_ik * B[k][j];
100 + C[i][j+1] += a_ik * B[k][j+1];
101 + C[i][j+2] += a_ik * B[k][j+2];
102 + C[i][j+3] += a_ik * B[k][j+3];
103 }
104 + // Handle remaining elements
105 + for(; j<n; j++) {
106 + C[i][j] += a_ik * B[k][j];
107 + }
108 }
109 }
110 }
```

● Changed.md を更新します：

EXPLORER

TEST

Mat-Mat-noopt\C

Makefile

mat-mat-noopt_v0_0.c

mat-mat-noopt_v1_0.c

mat-mat-noopt_v2_0.c

mat-mat-noopt_v3_0.c

mat-mat-noopt_v4_0.c

mat-mat-noopt_v5_0.c

mat-mat-noopt_v6_0.c

mat-mat-noopt_v7_0.c

mat-mat-noopt_v8_0.c

mat-mat-noopt.bash

mat-mat-noopt.c

Changed.md

mat-mat-noopt_v2_0.c

Makefile

mat-mat-noopt.bash

Changed.md

Preview Changed.md

- N=512での実行結果: 3.752357秒、71.537833 MFLOPS (ジョブID: 7496934)
- 結果: 性能向上！

Version 5.0 (ikjループ順序)

- ループ順序をikjに変更
- キャッシュ効率の向上を狙う
- N=512での実行結果: 2.639626秒、101.694501 MFLOPS (ジョブID: 7496935)
- 結果: 大幅な性能向上！オリジナルの約1.57倍

Version 6.0 (ikjループ順序 + jループ4段アンローリング)

- ikjループ順序でjループを4段アンローリング
- A[i][k]をレジスタに保持
- N=512での実行結果: 1.931697秒、138.963556 MFLOPS (ジョブID: 7496936)
- 結果: さらに性能向上！オリジナルの約2.15倍

Version 7.0 (ikjループ順序 + jループ8段アンローリング)

- ikjループ順序でjループを8段アンローリング
- N=512での実行結果: 1.819919秒、147.498544 MFLOPS (ジョブID: 7496937)
- 結果: 最高性能！オリジナルの約2.28倍

まとめ (N=512)

1. 最高性能: v7.0 (ikjループ順序 + j8段アンローリング) - 147.498544 MFLOPS
2. 効果的な最適化:
 - ループ順序の変更 (ijk→ikj): 約1.57倍の高速化
 - ikj順序でのjループアンローリング: さらに1.45倍の高速化
3. 効果が低かった最適化:
 - ijkループ順序でのk/jループアンローリング: 性能低下
 - iループアンローリング単体: 若干の改善のみ

推奨事項

- キャッシュ効率を考慮したループ順序の選択が最重要
- アンローリングは適切なループ順序と組み合わせることで効果を発揮
- 8段アンローリングが4段より若干良い結果を示した



Prototype of Multi-AI Agents (October 30, 2025 Version)

Multi-AI Agentization: Managing Multiple Automated Subordinate Agents



- Under **the guidance of HPC experts**, a code optimization development team is realized with **AI-assisted automation**.
- **Example Application Scenarios**
 - **Code optimization of CFD**
(modified **Himeno benchmark**)
 1. Parallelization with OpenMP
 2. Parallelization with MPI
 3. GPU acceleration using OpenACC
 4. GPU acceleration using CUDA



Multi-AI Agent Version: VibeCodeHPC Demo



CFD

Git-main

GitHub-main

Multi

EX01

VibeCodeHPC-jp-0.7.4

VibeCodeHPC-jp-0.7.5

_images

_remote_info

.cache

.claude

Agent-shared

BaseCode

communication

GitHub

hooks

instructions

telemetry

tmp

User-shared

.gitattributes

.gitignore

CLAUDE.md

README.md

requirement_definition_ex...

requirement_definition_te...

requirement_definition.md

requirements.txt

start_PM.sh

start_solo.sh

EX02

EX03

Solo

env_memo.md

要件定義書

プロジェクト情報

- プロジェクト名: CFD_Multi_EX1

最適化対象

- 非圧縮流体解析の圧力Poisson方程式をヤコビの反復法で解くコード
- 主にjacobi関数を最適化
- 配列初期化とテストコードは修正不要

対象ファイル

- BaseCode/以下を参照

最適化の度合い（目標）

性能目標

- 極限まで最適化を継続し、1ノードで最高性能を目指す

優先度

- ☒ スループット最大化
- ☐ スケーラビリティ

制約（指定）

meme@DESKTOP-UKQIT3: ~

Read 11 Lines (ctrl+o to expand)

Read_remote_info/Flow/node_resource_groups.md

Read 97 Lines (ctrl+o to expand)

Read_remote_info/Flow/sample_bash.sh

Read 18 Lines (ctrl+o to expand)

Thinking...

プロジェクト開始時刻を記録中... (esc to interrupt · ctrl+t to show todos · 125s · 4.1k tokens)

Next: 予算管理初期化 (開始時点の予算残額確認)

PM:プロンプト読み込み中...

(待機中0) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'PMが directory_pane_map.mdで配置を決定します'

(待機中0) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'その後、エージェントが起動されます'

(待機中0) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo ''

(待機中0) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'OpenTelemetryが有効化されています'

(待機中0) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'OTLPエンドポイント: http://localhost:4317'

(待機中0) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$

(待機中2) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'PMが directory_pane_map.mdで配置を決定します'

(待機中2) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'その後、エージェントが起動されます'

(待機中2) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo ''

(待機中2) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'OpenTelemetryが有効化されています'

(待機中2) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'OTLPエンドポイント: http://localhost:4317'

(待機中2) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$

echo 'その後、エージェントが起動されます'

(待機中4) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'PMが directory_pane_map.mdで配置を決定します'

(待機中4) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'その後、エージェントが起動されます'

(待機中4) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo ''

(待機中4) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'OpenTelemetryが有効化されています'

(待機中4) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'OTLPエンドポイント: http://localhost:4317'

(待機中4) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$

=== エージェント配置待ち (Pane 5) ===

(待機中5) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo ''

(待機中5) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'PMが directory_pane_map.mdで配置を決定します'

(待機中5) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'その後、エージェントが起動されます'

(待機中5) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo ''

(待機中5) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'OpenTelemetryが有効化されています'

(待機中5) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$ echo 'OTLPエンドポイント: http://localhost:4317'

(待機中5) /mnt/c/Users/memo/cc/CFD/Multi/EX01/VibeCodeHPC-jp-0.7.5\$

CFD_Multi0:hpc-agents*

"Pane3" 16:57 19-Oct-25

Report on Multi-AI Agent Version of VibeCodeHPC



- Automatically generate reports including SOTA (State-of-the-Art) performance, agent-specific effects, and insights into effective optimizations.

エージェント	役割	主な貢献
PM	プロジェクト管理	全体統括、予算管理、進化的探索戦略立案
SE1	システム設計	SOTA監視、統計分析、予算追跡
PG1.1	OpenMP実装	47.2 GFLOPS達成、SIMD最適化確立
PG1.2	MPI/ ハイブリッド	68.803 GFLOPS達成 (PROJECT SOTA)
PG1.3	CUDA実装	21.1 GFLOPS達成、GPU最適化探索
PG1.4	OpenACC実装	24.4 GFLOPS達成、ループタイリング最適化
CD	GitHub管理	継続的同期、成果物管理

7.2 協調作業の成果

- 定期報告: agent_send.shによる非同期通信
- SOTA共有: ChangeLog.md + sofa_checker.pyによる自動追跡
- 予算管理: budget_tracker.pyによるリアルタイム集計
- 知見共有: 各PGの失敗事例がプロジェクト全体の学習に貢献

8. プロジェクト管理の振り返り

8.1 成功要因

- 進化的探索戦略: Gen1単一技術 → Gen2ハイブリッドの段階的アプローチ
- Flat Directory構造: 並列探索の効率化、階層木の一意性問題を回避
- 自動化ツール: SOTA追跡、予算集計の自動化により迅速な意思決定
- マルチエージェント体制: 並列実装探索により65分で5技術+ハイブリッドを実証

CPU最適化
OpenMP
47.2GFLOPs

MPI(+OpenMP)
68.8GFLOPs

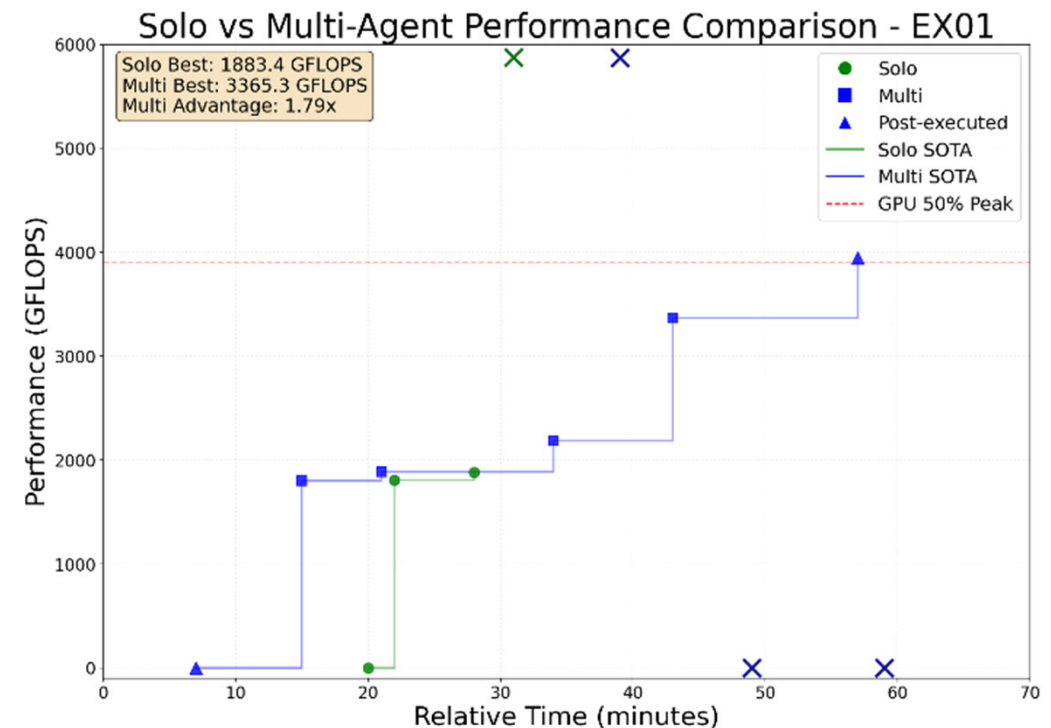


Case Study

- Multi-AI agents code opination for matrix-matrix multiplication

```
void gemm_naive (int M, int N, int K,  
                double alpha, const double* A, int lda,  
                const double* B, int ldb,  
                double beta, double* C, int ldc) {  
    for (int i = 0; i < M; i++) {  
        for (int j = 0; j < N; j++) {  
            double sum = 0.0;  
            for (int k = 0; k < K; k++) {  
                sum += A[i * lda + k] * B[k * ldb + j];  
            }  
            C[i * ldc + j] = alpha * sum + beta * C[i * ldc + j];  
        }  
    }  
}
```

Original Code



Optimization History

Technologies Used for Multi-Agent Implementation



Overview of Methods (*also included in final_report.md*) **v1.4.0 CUDA Optimization** (Theoretical Performance: 43.14%)

- **Double buffering** to hide L2 cache access latency
- Cache optimization using **__ldg** for read-only data

🚀 実装の技術的成果

🔗 成功した最適化手法

1. ダブルバッファリング (v1.4.0)

- 計算と転送のオーバーラップ
- 15.12%の性能向上 (28.02%→43.14%)

2. 共有メモリアイリング

- ブロックサイズ最適化 (16→32→64)
- メモリアクセスパターン改善

3. レジスタタイリング

- 4x4タイル実装
- レジスタ活用の最大化

4. read-onlyキャッシュ活用

- `__ldg()`によるキャッシュ最適化
- メモリ帯域の効率化

Limitations of a Solo Agent



- Frequently forget to push to GitHub (~80% of the time)
- Cannot fully perform the CD role
- Even when requirements specify testing both single and multi-agent setups, one is often neglected
- **Often forgets the restriction against using cuBLAS**

With multi-agent setups, agents can monitor each other (Example below shows a **CD**  **detecting a violation**)

```
247 [2025-09-15 16:43:02] PM: "[SE1] 【重要】v1.3.0で大幅性能向上! 5868.981 GFLOPS(75.24%)達成! cuBLAS+Tensor Core活用。目標60%を突破しました!"
248 [2025-09-15 16:43:03] PM: "[CD] 警告: v1.3.0でcuBLAS使用を検出! 要件定義で禁止されています。性能75.24%(5868.981 GFLOPS)は無効です"
249 [2025-09-15 16:43:08] SE1: "[PM] 重大成果! v1.3.0で75.24%達成(5868.981 GFLOPS)! cuBLAS+Tensor Core活用。統計更新をお願いします"
250 [2025-09-15 16:43:17] PG1.1: "[PM] 緊急停止! v1.3.0はcuBLAS使用のため要件違反です。自前実装のみで最適化してください。cuBLASなしで60%達成が目標です"
251 [2025-09-15 16:43:24] SE1: "[PM] v1.3.0は無効。cuBLAS使用は要件違反です。グラフから除外してください。有効な最高性能は v1.2.1の28.02%です"
252 [2025-09-15 16:43:35] PM: "[SE1] 了解しました。v1.3.0を無効として処理します。有効な最高性能はv1.2.1の28.02%です。グラフを更新します。"
```



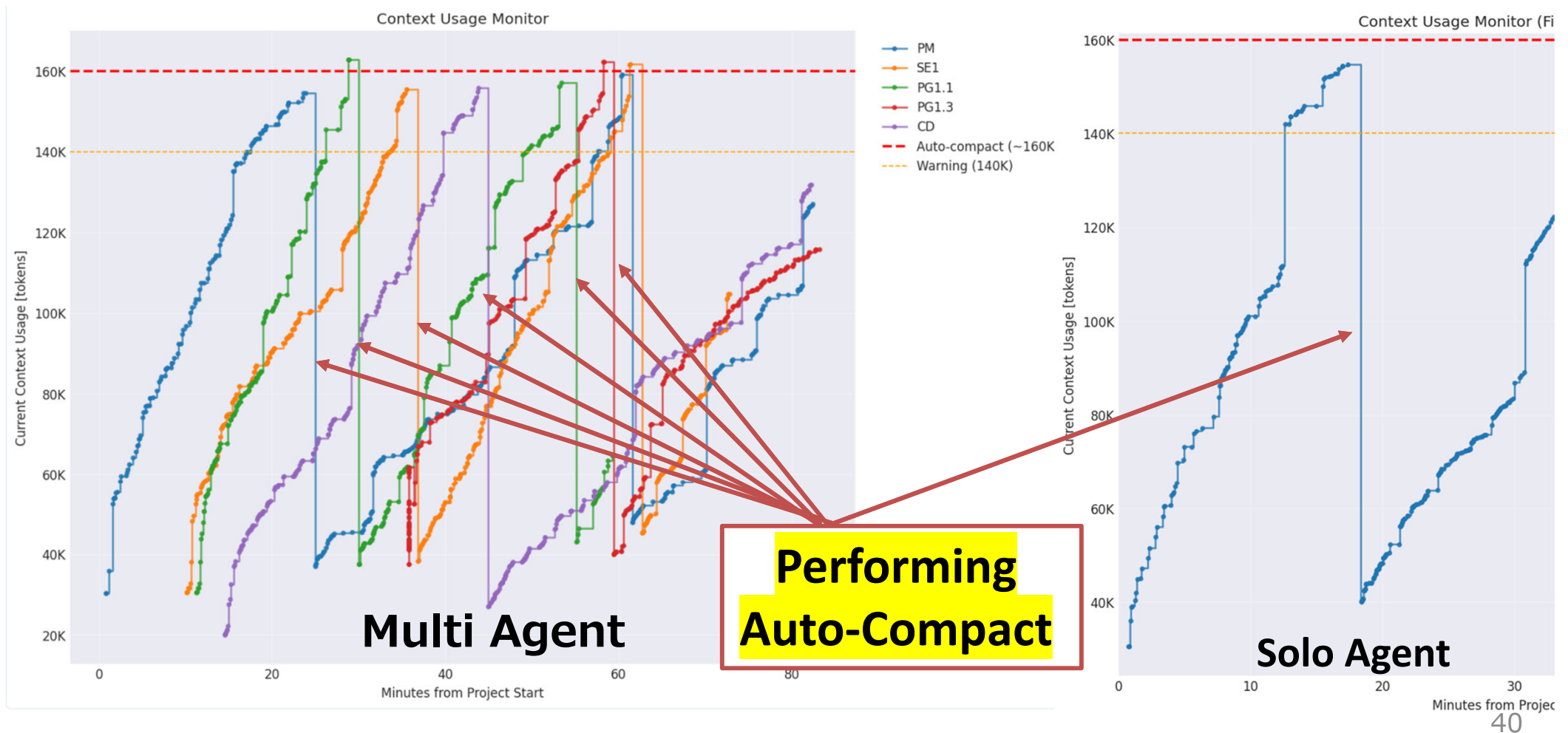
Monitoring of Context Consumption Over Time [tokens]

Claude's Context Length [tokens] time transition



Implemented functionality for the PM to dynamically launch agents.

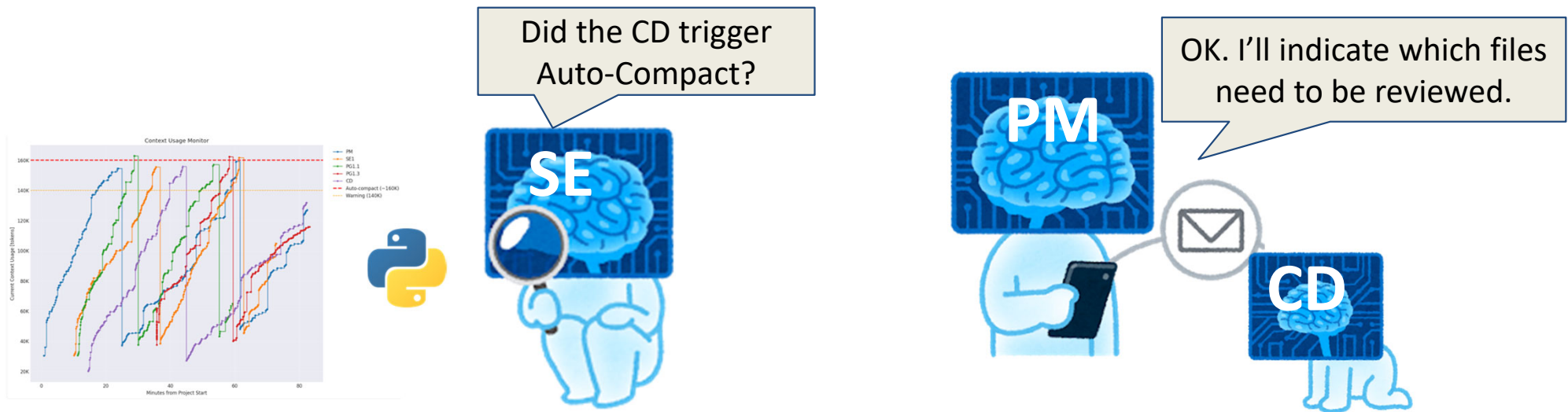
Developed a system where PM and SE can monitor PG's token consumption in real time.



Resolved post-processing issues following multi-agent auto-compaction (memory compression)



Performing Auto-Compact causes active memory to degenerate → resulting in loss of knowledge.



Translated Messages:

To SE1: `` `[PM Overview] PG2 successfully submitted with Job ID: 2053140. budget_tracker recognizes 2 running jobs (PG1 may also have been submitted). CD context remains at 96.3%—stay alert. Please prioritize countermeasures for auto-compact. ``

To PM: `` `[SE1 Report] 30-minute snapshot completed. All graphs saved to User-shared/snapshots/30min/. PG2 Job 2053140 is running. budget_tracker recognizes 2 running jobs. CD context improved to 20.3% (Auto-compact may have occurred?). ``

To CD: `` `[PM] CD context improved to 20.3%! Auto-compact may have occurred. Please reload CLAUDE.md and directory_pane_map.md as needed. Report GitHub synchronization status. ``

Closing Remarks



- With advances in code-generation AI, the progress in automated code tuning using multi-AI agents has been remarkable.
- We have developed **VibeCodeHPC**, a multi-agent code-generation system based on Vibe Coding.

Future Work

- Using more complex, practical software:
 - Performance evaluation
 - Methodological research on automated code generation
- Copyright and safety considerations of auto-generated code
- Introduction of software engineering research practices
 - Assessment of code maintainability (readability, maintainability, continuous operation, etc.)