



HPC-GENIE Project HP:
https://www.hpc.itc.nagoya-u.ac.jp/menu/hpc_genie.html



HPC-GENIE: A Multi-Agent Code Generation Platform Project Based on Context Engineering

Takahiro Katagiri
Nagoya University

JHPCN Field Workshop

State-of-the-Art in Code Generative AI for High-Performance Computing

December 5th (Fri), 2025, 15:30 - 16:00

Lecture Room, 2F, Information Technology Center, Nagoya University

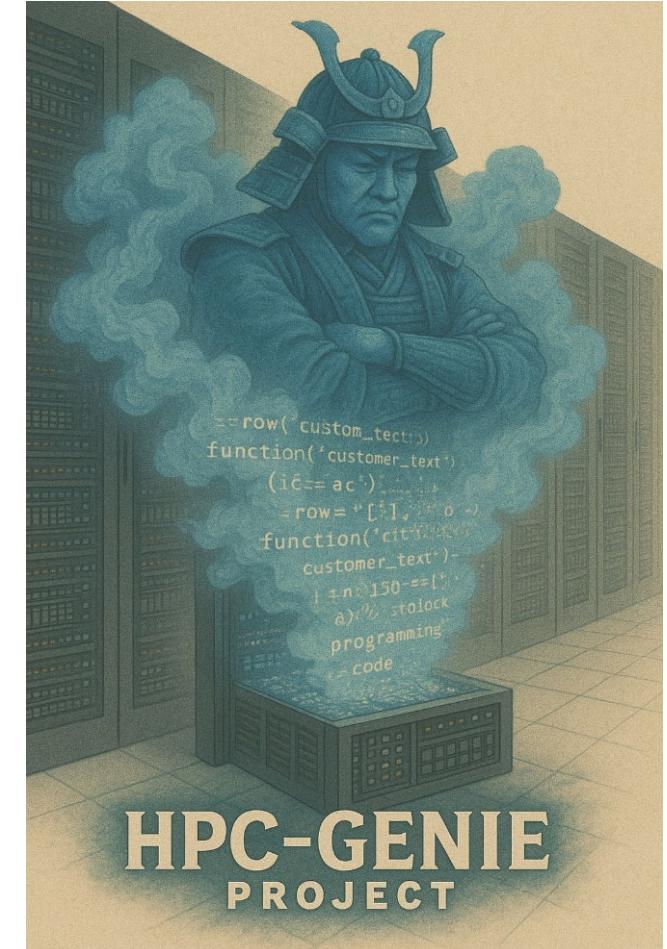
HPC-GENIE Project

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-GENIE
PROJECT

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



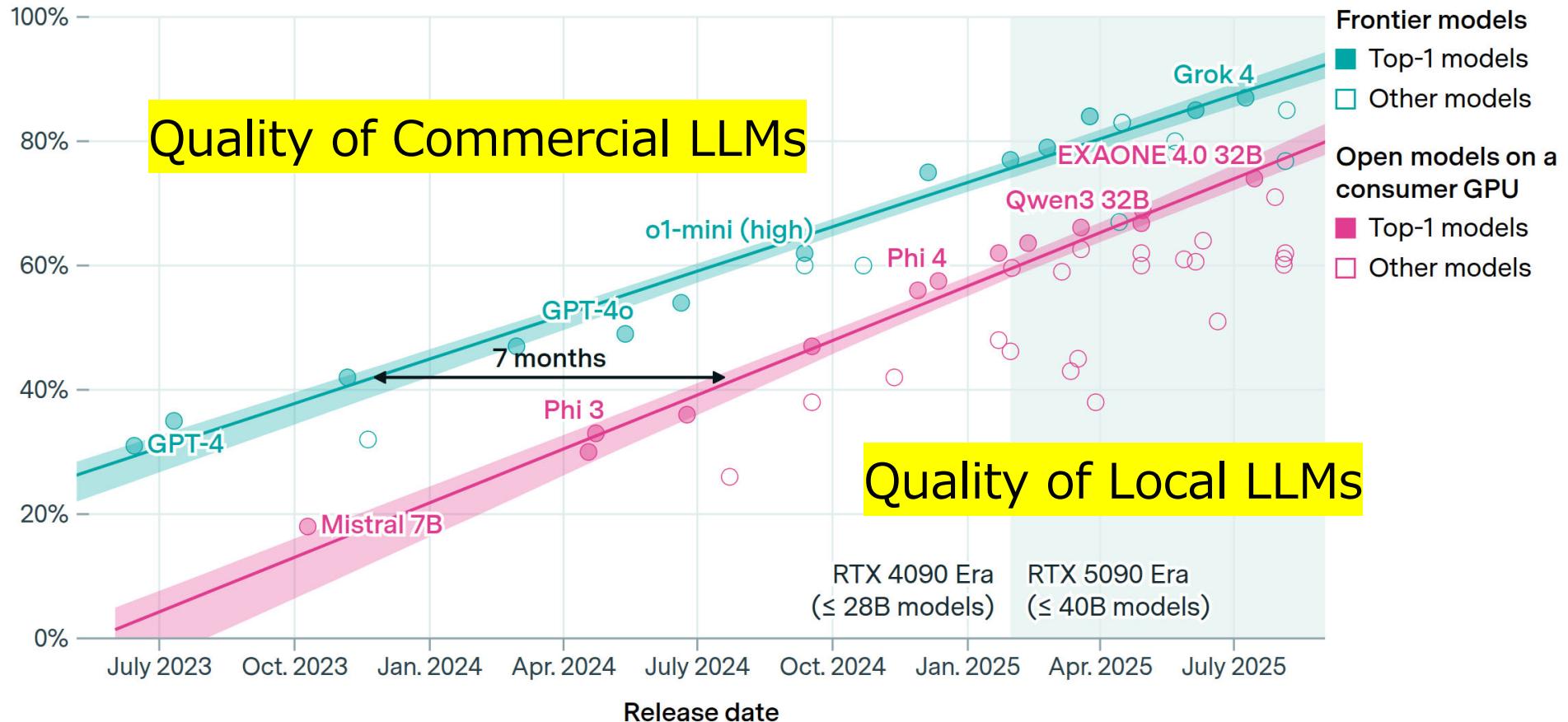
Item	Conventional LLMs	Focus on HPC-GENIE Project
Code Generation Method	Manual prompt input to a general-purpose LLM and use of its responses	Prompt engineering combined with a CLI-based multi-LLM selection approach
Integration with automatic tuning	Manual connection via CLI tools (e.g. OpenTuner, GPTune)	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

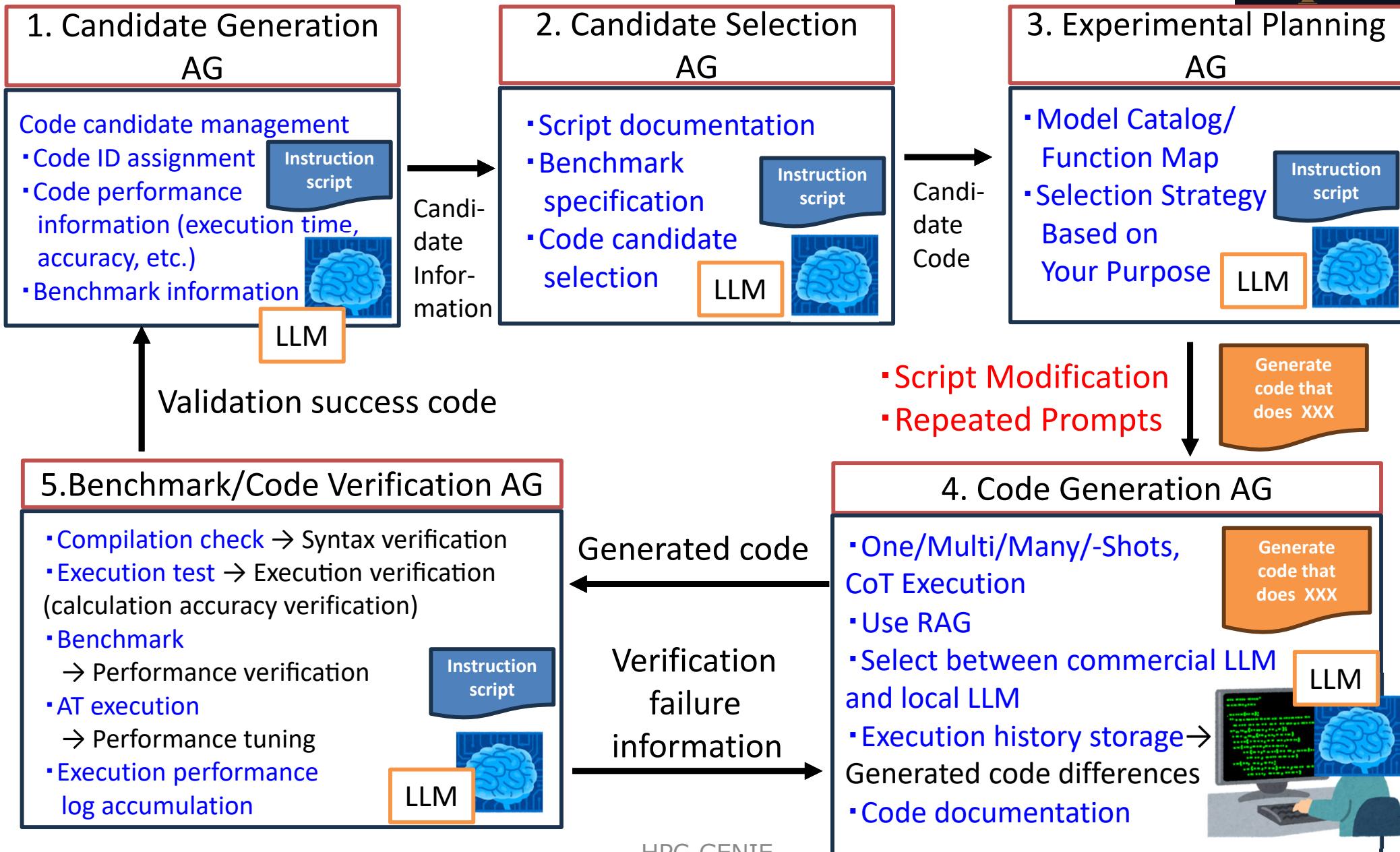


CC-BY

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

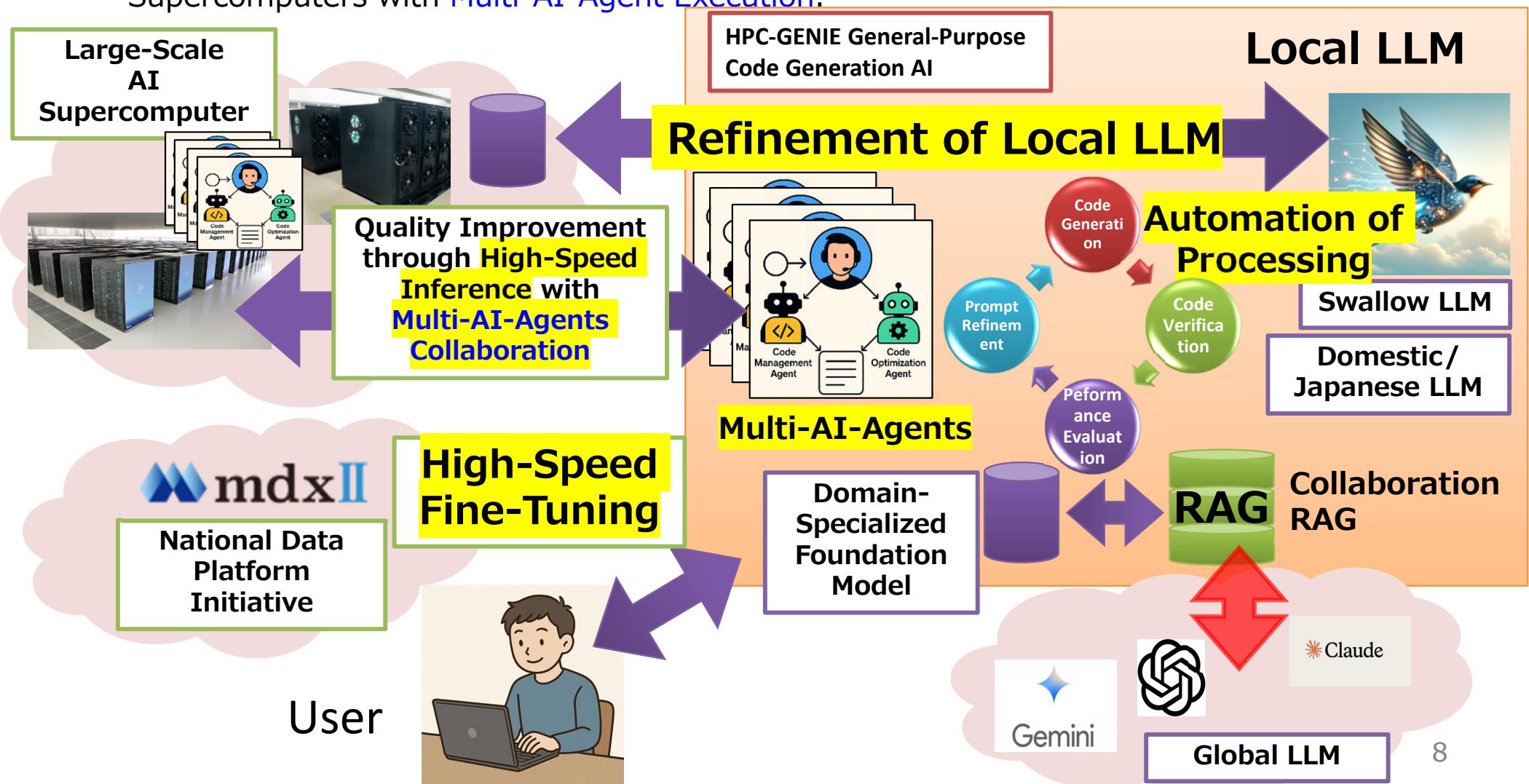
Iterative Prompting and AI Agents



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



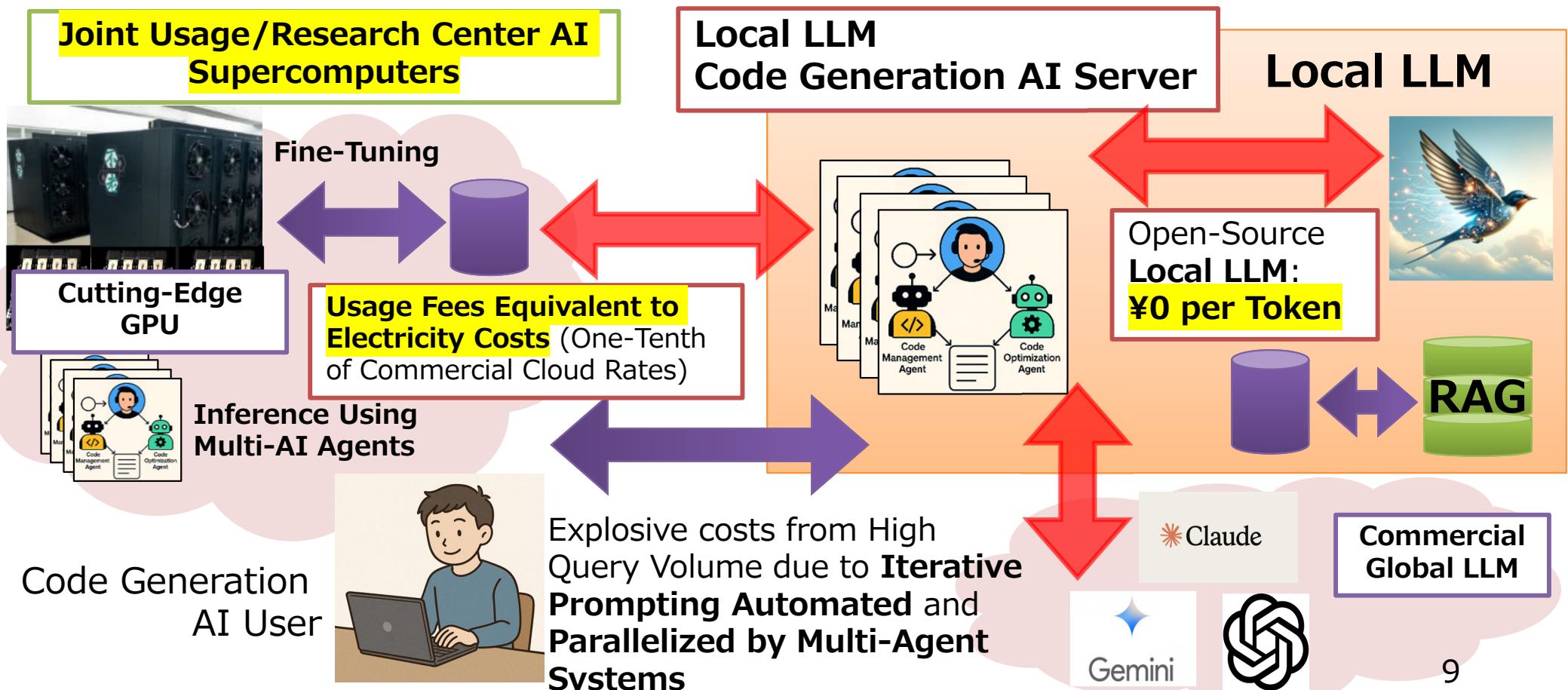
- **Explosive Costs from Using Commercial LLMs**

- 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.



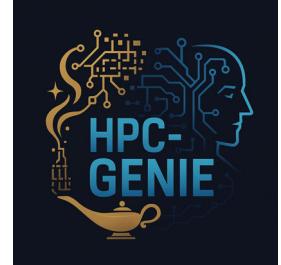


VibeCodeHPC

: 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 Cording 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 **2.8x**.
- **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 2.8x**.
- **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× 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×**.
- **Developer:** Good!



VibeCodeHPC

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

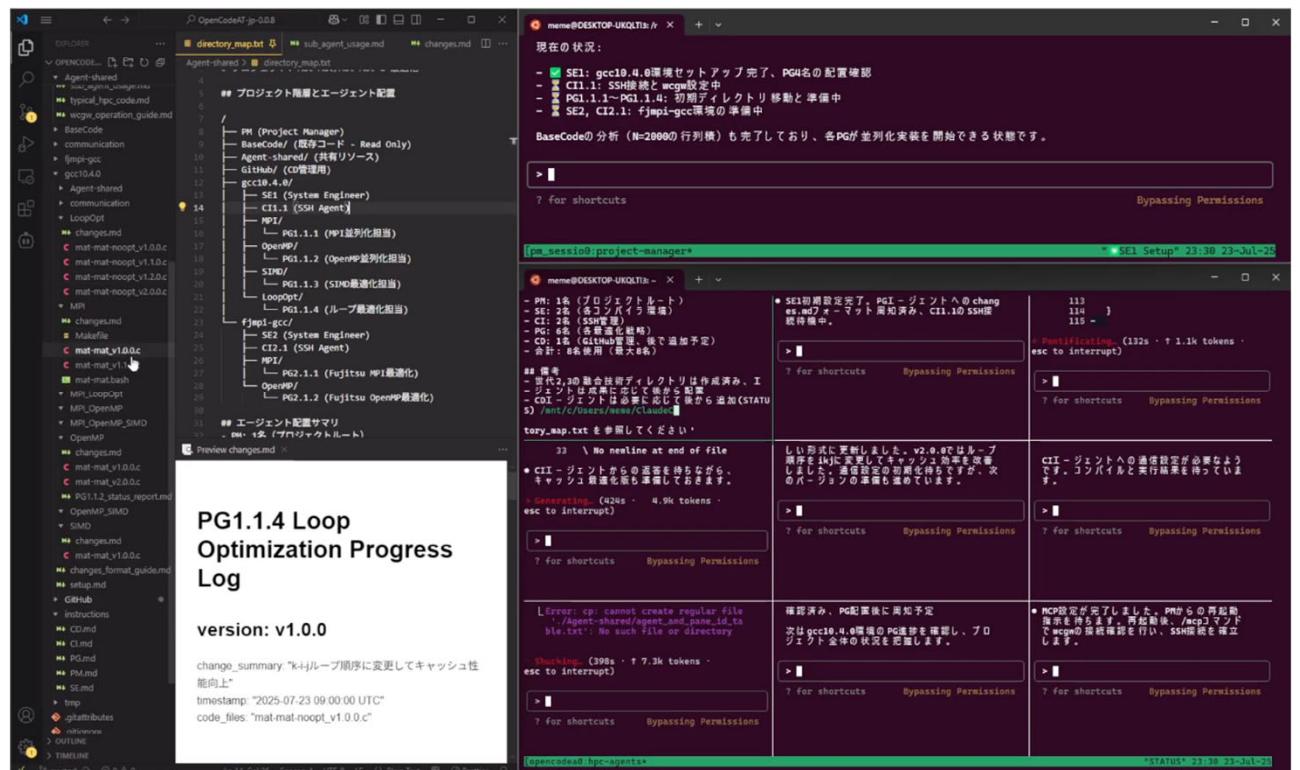
Main Developer:
Shun-ichiro Hayashi
(M1, Graduate School of Informatics, Nagoya University)

Overview of VibeCodeHPC (1/3)



- **VibeCodeHPC**
 - Multi Agentic Vibe Coding for HPC
- Released on GitHub on July 28, 2025

GitHub :
[https://github.com/](https://github.com/Katagiri-Hoshino-Lab/VibeCodeHPC-jp)
[Katagiri-Hoshino-
Lab/VibeCodeHPC-jp](https://github.com/Katagiri-Hoshino-Lab/VibeCodeHPC-jp)



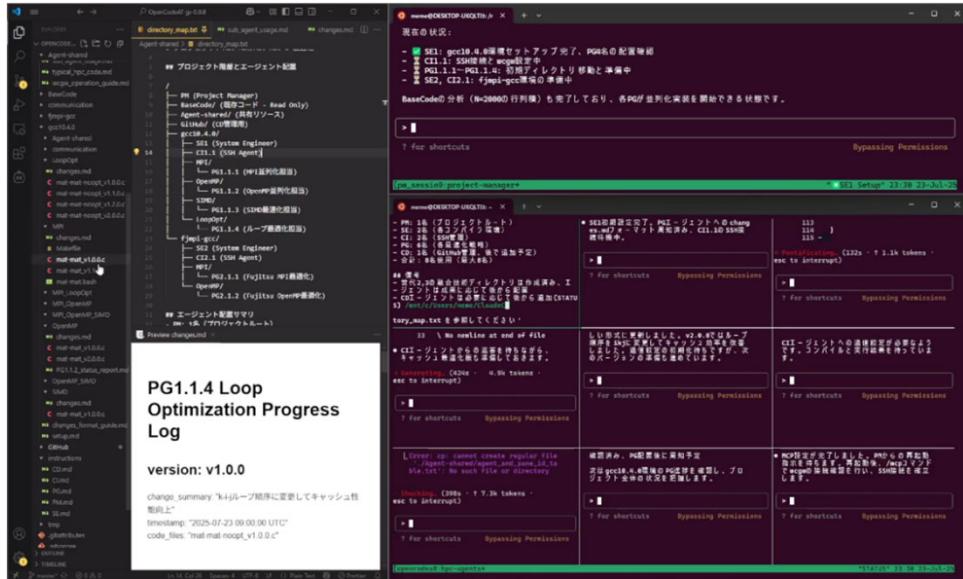


Overview of VibeCodeHPC (2/3)

- 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

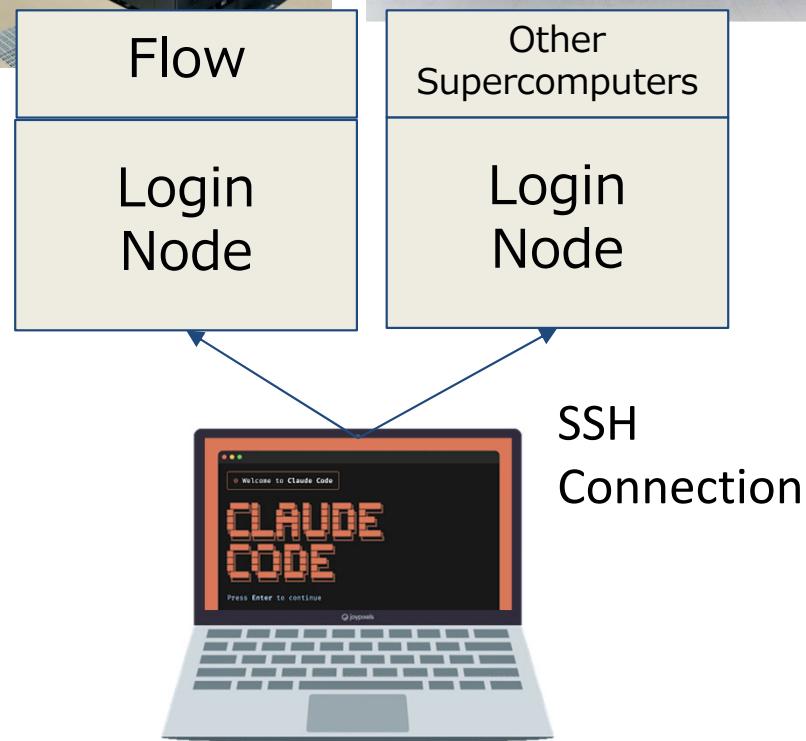
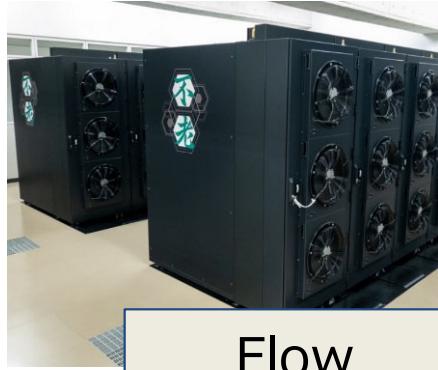


Early Prototype Version

(July 28, 2025)

Single-Agent Edition

Implementation for Supercomputer Utilization



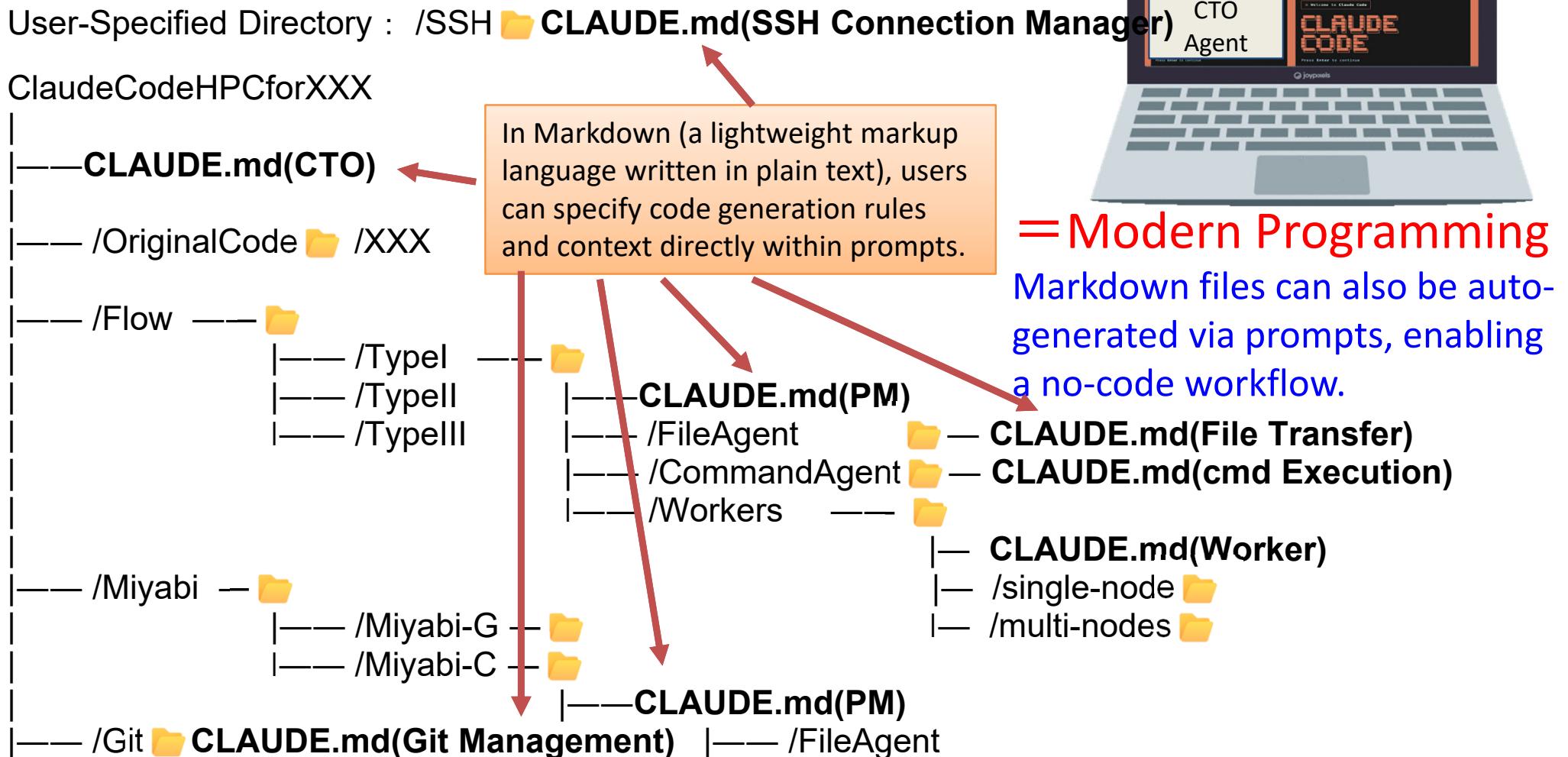
Running Claude Code on a Local PC

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

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



=Modern Programming

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

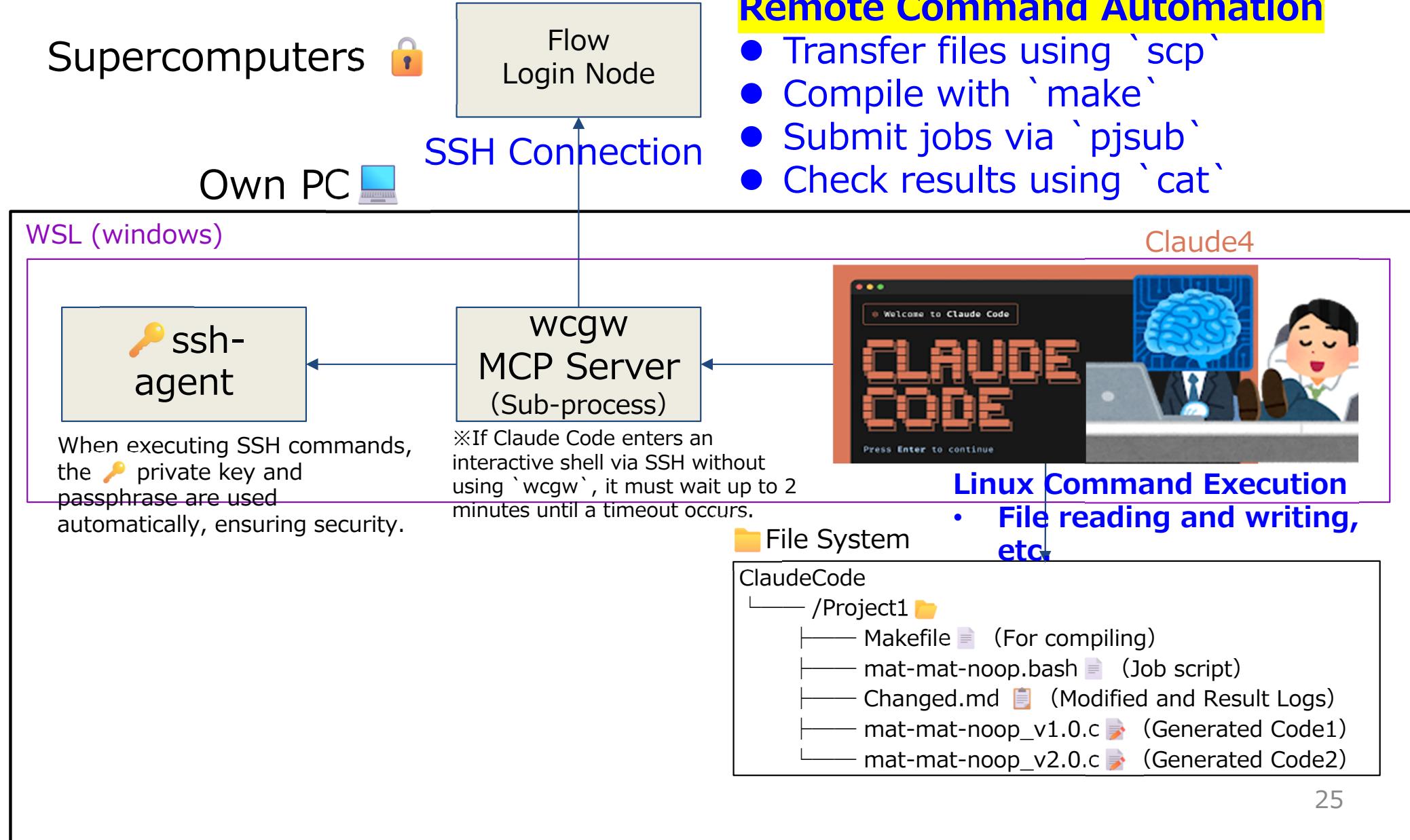
CLAUDE.md(File Transfer)
CLAUDE.md(cmd Execution)

|— CLAUDE.md(Worker)

single node

|- single-node

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



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 l → \$ make → submit the job with pbsub → 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 \$ pbsub 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 × N double-precision matrices A and B, storing the result in C.

Automatically Generated Report Upon Optimization Completion



mat-mat-noopt.bash.7496935.out'\"}, wait_for_se
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++) {  
98                 c[i][j] += A[i][k] * B[k][j];  
99             a_ik = A[i][k];  
100            for(j=0; j<n-3; j+=4) {  
101                c[i][j] += a_ik * B[k][j];  
102                c[i][j+1] += a_ik * B[k][j+1];  
103                c[i][j+2] += a_ik * B[k][j+2];  
104                c[i][j+3] += a_ik * B[k][j+3];  
105            }  
106            // Handle remaining elements  
107            for(; j<n; j++) {  
108                c[i][j] += a_ik * B[k][j];  
109            }  
110        }  
111    }  
112  
113    // ...  
114  
115    // ...  
116  
117    // ...  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1859  
1860  
1861  
1862  
1863
```



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



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.



terminal window showing MPI and OpenMP performance metrics. The window has tabs for 'ChangeLog.md', 'VIBE-OpenMP', 'final_report.md', and 'Preview final_report.md'. The main content displays large text overlays: 'CPU最適化 OpenMP 47.2GFLOPs' and 'MPI(+OpenMP) 68.8GFLOPs'. The terminal background shows command-line history related to MPI and OpenMP benchmarks.

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、進化の探索戦略立案
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

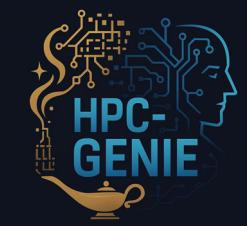
7.2 協調作業の成果

- 定期報告 `agent_send.sh`による非同期通信
- SOTA共有: `ChangeLog.md` + `sota_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

- 進化の探索戦略: 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

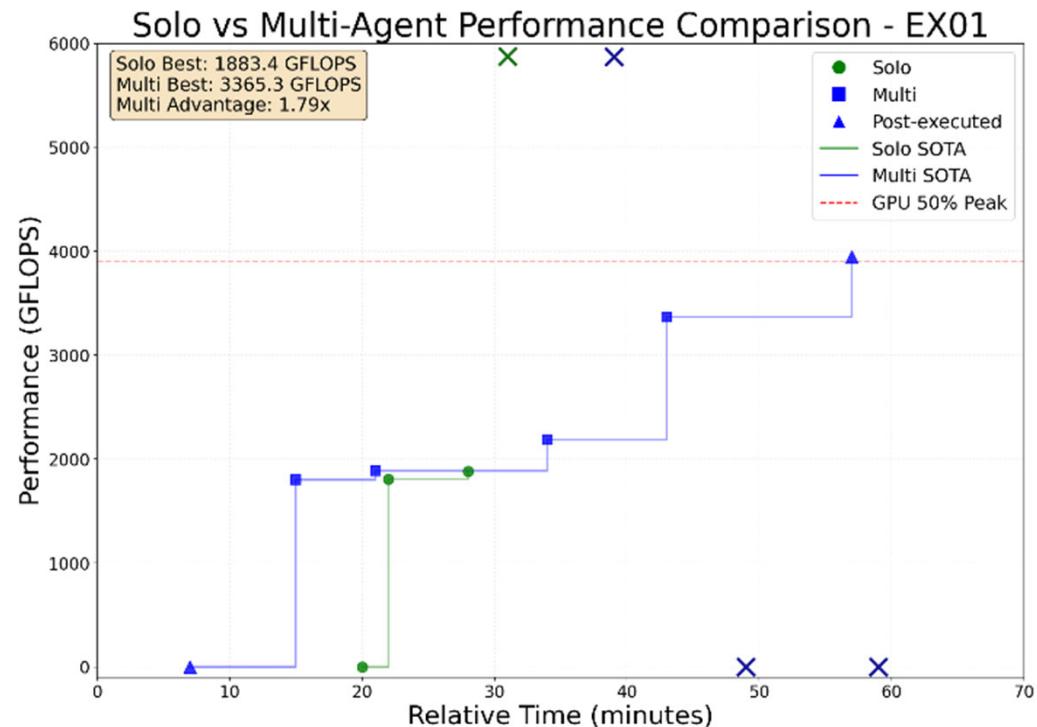


Case Study

- 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. 共有メモリタイリング
 - 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**

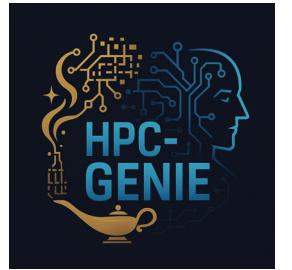
With multi-agent setups, agents can monitor each other
(Example below shows a **CD 🤖 detecting a violation**)

```
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。"
```



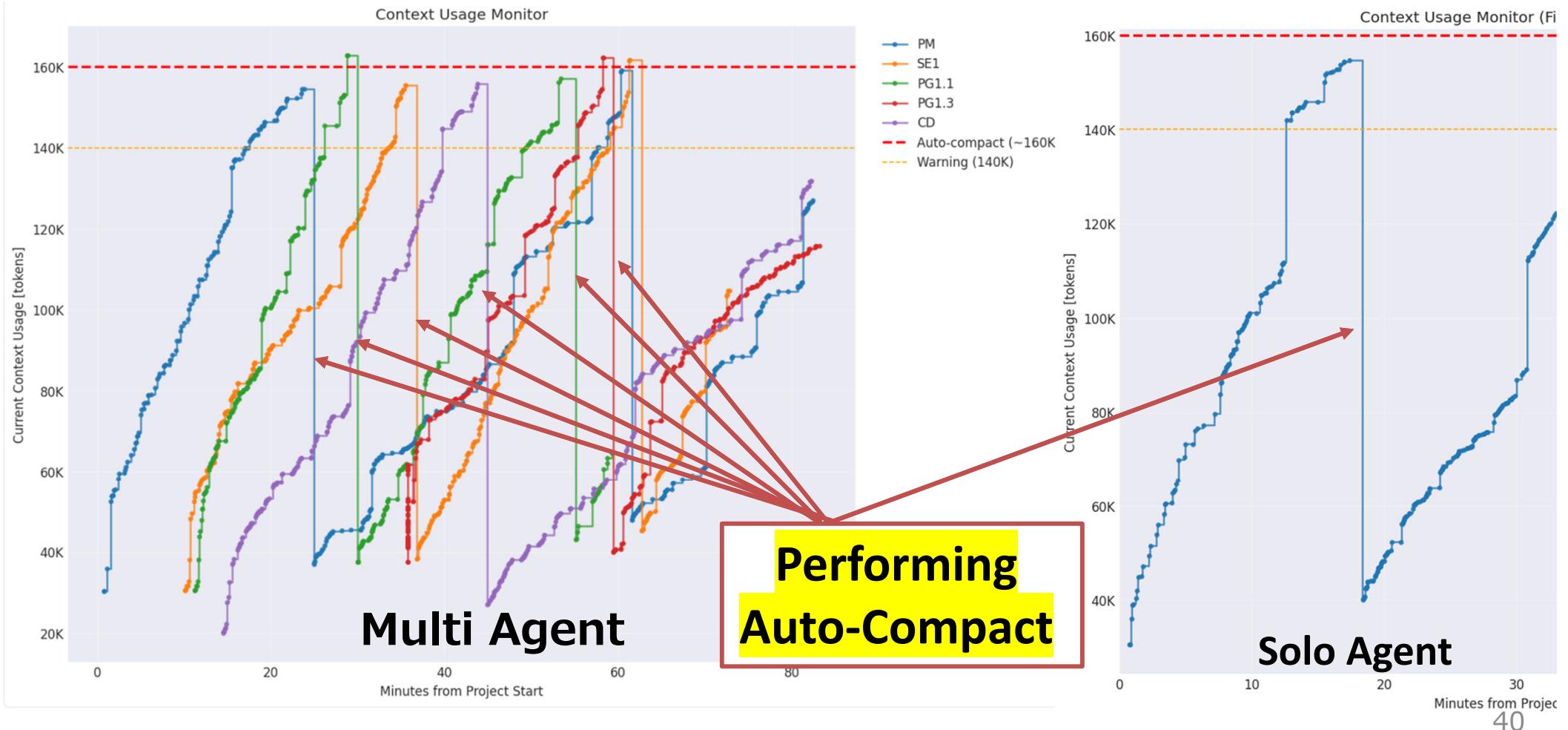
Monitering of Context Consumption Over Time [tokens]

Claude's Context Length [tokens] time transition



Implemented functionality for the PM to dynamically launch agents.

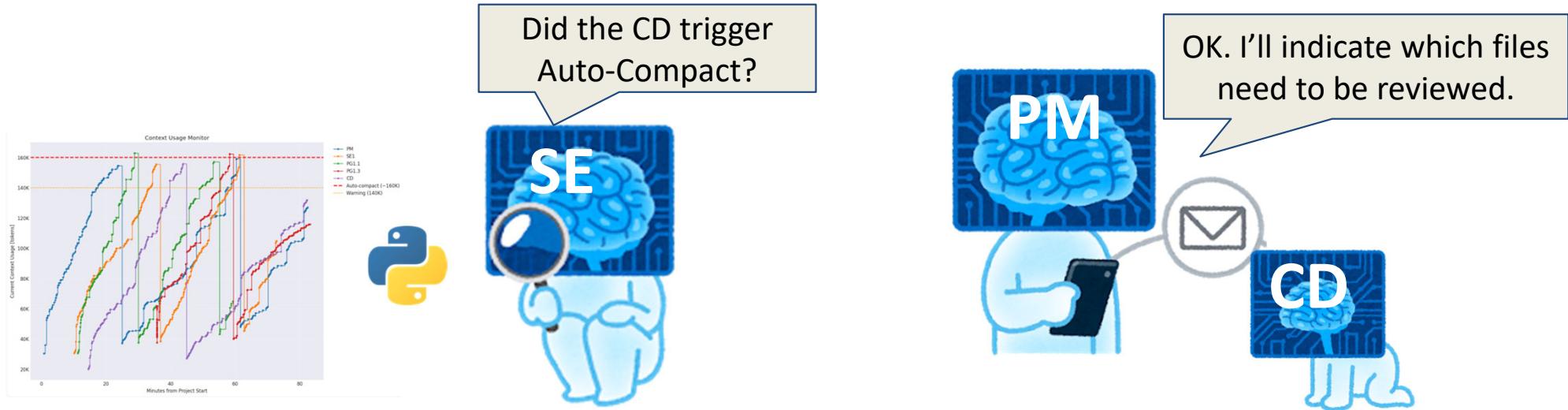
Developed a system where PM and SE can monitor PG's token consumption in real time.



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.)