HPC-GENIE Project
Generative AI for HPC Code Development

Daichi Mukunoki

Asst. Prof., Information Technology Center, Nagoya University, Japan

Collaborators: S. Hayashi (M2), K. Morita (M2), T. Kotama (M1), S. Sakaguchi (M1), R. Mikasa (M1), T. Hoshino (Assoc. Prof.), T. Katagiri (Prof.)

Japan–France Joint Workshop on LLM-Driven Code Generation and Automation for HPC and Scientific Computing
July 6, 2026, LIP6, Sorbonne University, Paris, France

Background: AI Coding

Background

- **AI coding is now the norm**: the shift to agentic systems gives models not just knowledge but skills
- Claude Code, Codex, Gemini CLI, ...

Evolution of coding agents (2025–)

- **Feb 2025** Vibe coding (by Karpathy): coding through natural-language dialogue
- **May 2025** Claude Code (Claude 4): CLI coding agents go mainstream
- **Mid 2025** Context engineering: designing what to put in context at each inference step
- **Summer 2025** Multi-agent: role-divided agents collaborate
- **Summer–Fall 2025** Spec-driven development (the spec is the program) & Skills (reusable packaged instructions/scripts that extend an agent)
- **2026–** Agentic engineering / harness engineering: designing environment, CI, tests, loops

From "how to ask" → "what to put in context" → "designing the execution infrastructure"

Background: Generative AI and HPC

Trends in generative-AI research for HPC

- **Evaluation & benchmarks**: assessing OpenMP/MPI/CUDA generation, ParEval (Nichols+ 2024), KernelBench (Ouyang+ 2025), etc. → "generation works, but correctness & performance guarantees are the key"
- **HPC-specific models, tools & frameworks**: OMPGPT (Chen+ 2024), HPC-GPT (Ding+ 2023), HPC-Coder (Nichols+ 2024), ChatBLAS (Valero-Lara+ 2024), ChatMPI (Valero-Lara+ 2026), legacy porting Fortran→C/C++ (Valero-Lara+ 2025), CodeRosetta (TehraniJamsaz+ 2024)
  - **ChatHPC** (Yin+ 2025): an end-to-end HPC framework integrating fine-tuning, tools & deployment
- **Agents / multi-agent**: MARCO (Rahman+ 2025), ASTRA (Wei+ 2025), ParaCodex (Kaplan+ 2026)
  - **VibeCodeHPC** (Hayashi+ 2026): our supercomputer-specialized multi-agent

Generative AI now covers the whole HPC code lifecycle: generation → optimization → training → operation

The HPC-GENIE Project (Nagoya University)

HPC-GENIE Project: Nagoya University (Apr 2025–)[a]

- **H**igh-**P**erformance **C**omputing with **GEN**erative Neural **I**ntelligence for **E**xecution
- Leading Japan in **applying generative AI to HPC code development**
- **We are HPC people, AI beginners** – AI-geek students are driving the project

Distinctive features

- **Agent development**: complementing model capabilities
- **Local LLMs**
  - Reduce dependence on commercial services – avoid lock-in, cut cost
  - Security – prevent leakage of code and data
  - Open / domestic technology, economic security
- Application to semiconductor development: JST project

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

Starting Point: Performance Evaluation (GPT-4.1/o4-mini)

BLAS generation with GPT-4.1/o4-mini[1)]

- **Generating working code from just the routine name** (zero-shot, pass@10)
- Structure differs from publicly available code (tends to minimize code) – the LLM understands the spec

SYCL code generation for Intel GPUs with Claude Code[2)]

- Porting CPU BLAS code to Intel GPUs via SYCL
- Works beyond the mainstream NVIDIA GPU (CUDA)

AI-based code porting – a future where AI becomes the compiler.
Hope for avoiding hardware vendor lock-in

[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.

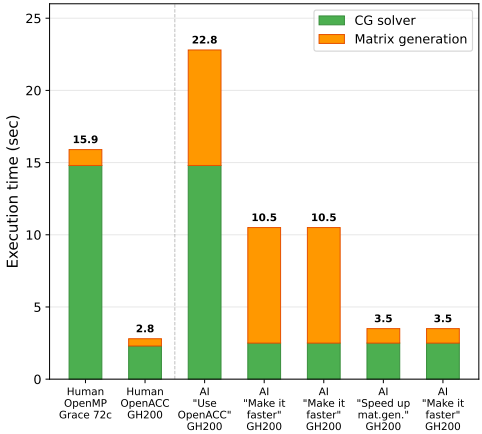
[2)] K. Morita et al., Evaluation of the Capability of Coding AI in Generating SYCL-Based Numerical Computation Codes for Intel GPUs, SCA/HPCAsia2026, poster session, Jan. 2026.

GPU Porting of Fortran Code (Opus 4.1)

GPU porting of FEM code with Claude Code [a]

- Porting a **Fortran 90** FEM code (MPI+OpenMP) to GPU with Claude Code (Opus 4.1) (target: GH200)
- Main components: coefficient-matrix generation + CG solver
- CG solver (typical) → **easily succeeds**
- Coefficient-matrix generation (custom code) → **often fails**

Good at typical code, weak at custom logic.
"Make it faster" prompts are effective



Optimization progress (y-axis: exec. time)
Leftmost: original, 2nd: manual opt.
3rd–right: AI opt.

[a] T. 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.

GPU Porting of Fortran Code (Opus 4.5 etc.)

GPU porting of an ICTCG-method code with three vendors' CLI agents [3)]

- A **Fortran 90** sparse-matrix solver code (OpenMP, ∼4000 lines)
- Comparing Claude Code (Opus 4.5), Codex (gpt-5.2-codex), Gemini CLI (gemini-2.5-pro)
- Introducing **deoptimization**: removing CPU-specific optimizations

GPU porting result (speedup vs. CPU)

Input code state	Claude	Codex	Gemini
Original (CPU-optimized)	**2.6**×	fail	fail
Stage-1 (remove macros/comments)	–	fail	fail
Stage-2 (remove OpenMP → serialized)	–	**6.3**×	fail

Deoptimization improves AI's GPU-porting success rate.
Keep code clean → AI optimizes for each hardware, like a compiler

[3)] S. Sakaguchi et al., Comparison of code-generation AIs for GPU porting of an ICTCG-method computation program, IPSJ 88th National Convention, Mar. 2026.

VibeCodeHPC: Multi-Agent HPC Code Optimization (Summer 2025)

The era of AI agents and multi-agent systems

- Following the spread of single CLI agents (Claude Code, etc.), since 2025 **multi-agent** systems (coordination of role-divided agents) are the new trend
- **Pros**: distribute context windows, explore diverse strategies in parallel, mutual monitoring / **Cons**: higher context consumption, agent design

VibeCodeHPC[4)5)] – a supercomputer-specialized multi-agent for HPC code optimization

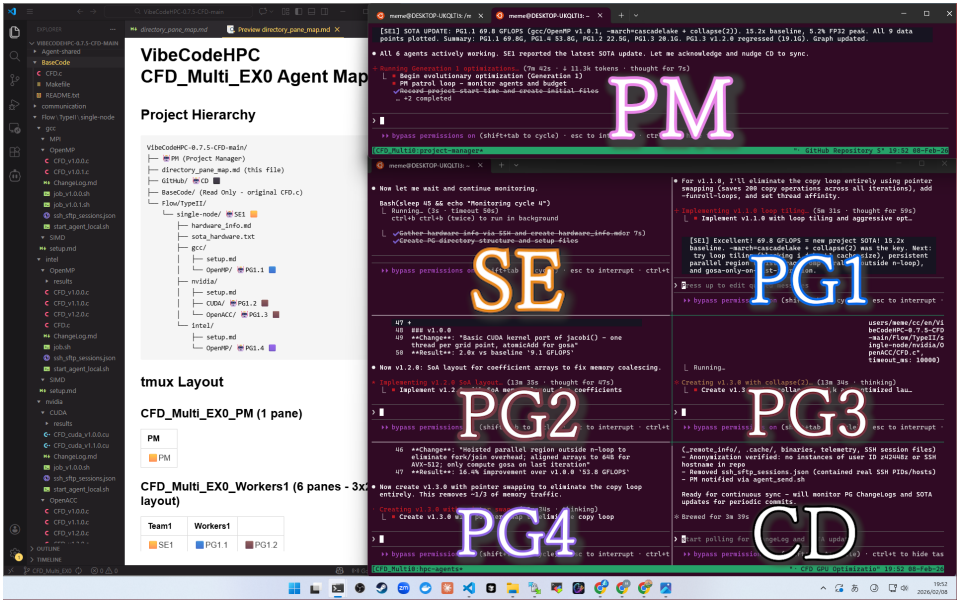
- Built on Claude Code (now supports any CLI tool). Roles = PM / SE / PG / CD
- Communicates via tmux, dynamically places agents, operates various supercomputers. **Supports not only Vibe Coding but also long, self-running operation**
- **Agents watch each other** to catch rule violations
- **Tracks each agent's context use** and cleans up its memory before it overflows = the key to long operation

Developed in summer 2025, the dawn of multi-agent systems (main developer: S. Hayashi, 2nd-year master's student)

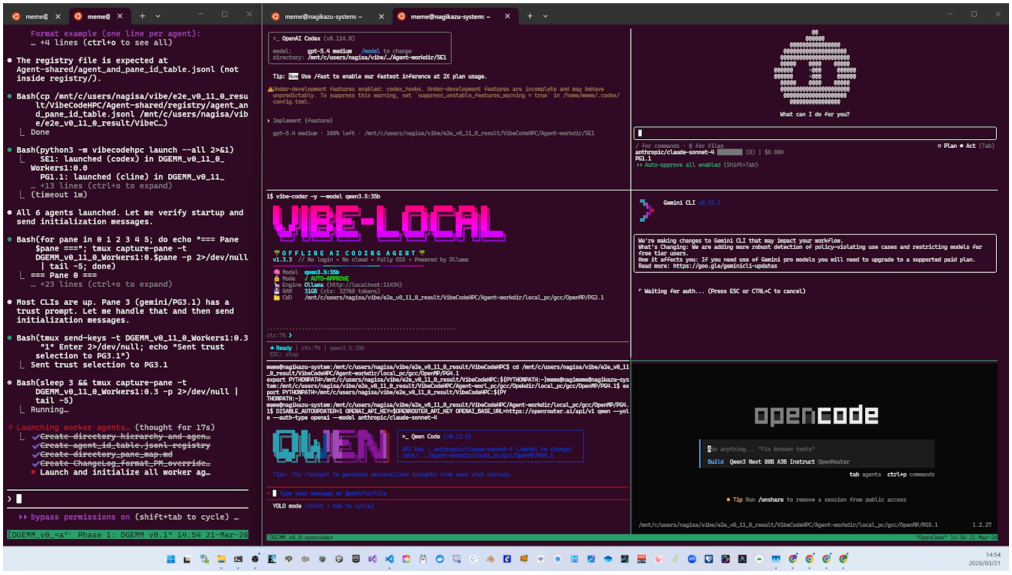
[4)] https://github.com/Katagiri-Hoshino-Lab/VibeCodeHPC-jp

[5)] S. Hayashi et al., VibeCodeHPC: A Multi LLM Agent System for HPC Code Auto-Tuning, iWAPT 2026 in IPDPS 2026.

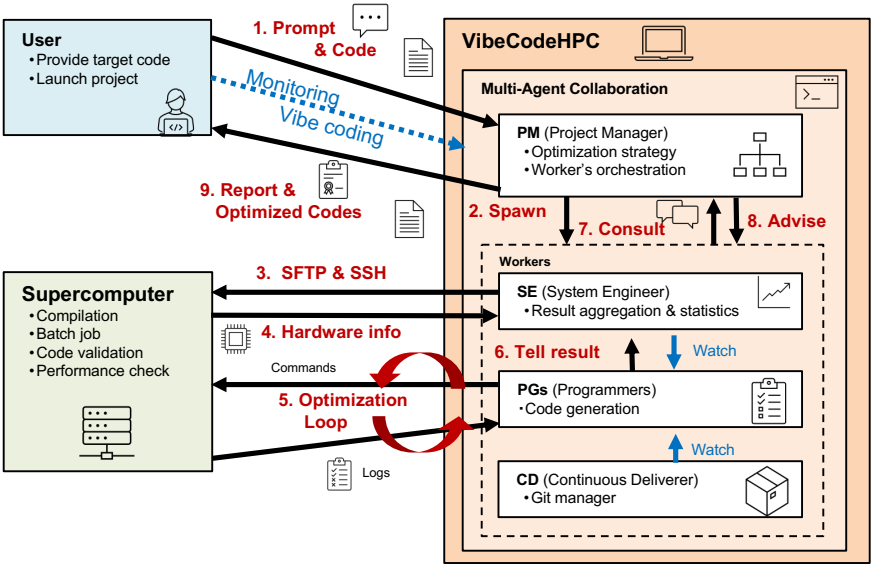
VibeCodeHPC – Demo Video



VibeCodeHPC – Multi-CLI Support



VibeCodeHPC – Workflow

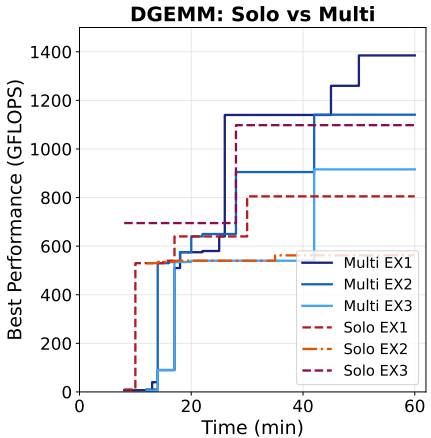


VibeCodeHPC – Matrix-Multiply Optimization on a GPU Supercomputer

Performance (GFlops/s, n=2048)

Tech	Mode	Run 1	Run 2	Run 3
OpenMP	Multi	105	213	190
	Solo	–	–	–
MPI	Multi	7	31	–
SIMD	Multi	–	184	–
CUDA	Multi	**1385**	**1141**	916
	Solo	805	562	**1098**
OpenACC	Multi	250	–	–
MPI+CUDA	Multi	–	–	332

Xeon Gold 6230 (20c) ×2 + Tesla V100 ×4



Multi-agent explores more strategies, reaching higher performance while still following the task rules

Automating HPC Research (HPC-AutoResearch)

Prior work: The AI Scientist[a] (Sakana AI 2025)

- **Research-automation** multi-agent system: idea → implementation → experiment → evaluation → paper
- Works with any backend LLM; specialized for AI research using Python

Adapting to HPC research – HPC-AutoResearch[b]

- A split-phase execution model handles trial-and-error setup/experiments on HPC systems
- Structured memory manages knowledge across research phases
- Generates papers surpassing commercial AI agents
- Successor: **ARI** (Autonomous Research Infrastructure)

[a] M.J. Fontaine et al, The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery, arXiv2412.05210, 2025

[b] T. Kotama et al., "HPC-AutoResearch: Adapting Autonomous Research Systems for HPC through Split-Phase Execution," GeCoin in IJCAI-ECAI 2026, Jun. 2026 (accepted).
`https://github.com/kotama7/HPC-AutoResearch`

LLM-review scores (max 4)

Criterion	Prop.	No-mem	Claude	Codex	Gemini
Novelty	**3**	3	2	2	2
Quality	**3**	3	3	1	2
Clarity	**3**	3	3	2	2
Significance	**3**	3	2	2	3
Soundness	**3**	3	3	1	3
Presentation	**3**	3	3	2	2
Contribution	**3**	3	2	1	3
Overall (/10)	7	7	6	3	7
Decision	A	A	A	R	A
Node success	**47.0**	40.5	–	–	–

"No-mem" = ablation removing memory. Review scores identical, but node-success drops 47.0→40.5% (memory stabilizes code generation). Decision: A=Accept, R=Reject

Research-Automation Agent – ARI

ARI: Autonomous Research Infrastructure[a]

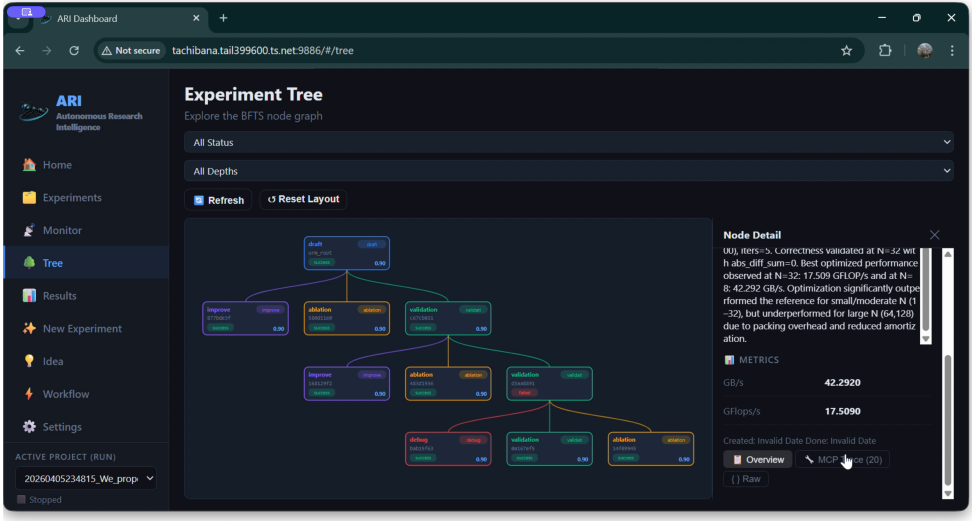
- **Successor** to HPC-AutoResearch. Handles any topic that uses computers
- **Just write the research topic in Markdown**: from a few lines, it autonomously runs ideation → experiment → verification → paper
- ARI's novelty: an **evidence-driven search over hypotheses**, a reasoning-and-action loop, and **reusable skills** → results + paper + reproducibility report

Not only coding – automating the whole research process is beginning (main developer: T. Kotama, 1st-year master's student)

[a] ARI – Autonomous Research Infrastructure, `https://github.com/kotama7/ARI`

A generated paper: a CSR-SpMM study on aarch64

ARI – Demo


ARI Dashboard
Not secure
tachibana.tail399600.ts.net:9886/#/tree
ARI
Autonomous Research Intelligence
Home
Experiments
Monitor
Tree
Results
New Experiment
Idea
Workflow
Settings
ACTIVE PROJECT (RUN)
20260405234815_We_prop
Stopped
Experiment Tree
Explore the BFTS node graph
All Status
All Depths
Refresh
Reset Layout
Node Detail
00), iters=5. Correctness validated at N=32 wit
h abs_diff_sum=0. Best optimized performance
observed at N=32: 17.509 GFLOP/s and at N=
8: 42.292 GB/s. Optimization significantly outpe
rformed the reference for small/moderate N (1
−32), but underperformed for large N (64,128)
due to packing overhead and reduced amortiz
ation.
METRICS
GB/s
42.2920
GFlops/s
17.5090
Created: Invalid Date Done: Invalid Date
Overview
{ } Raw

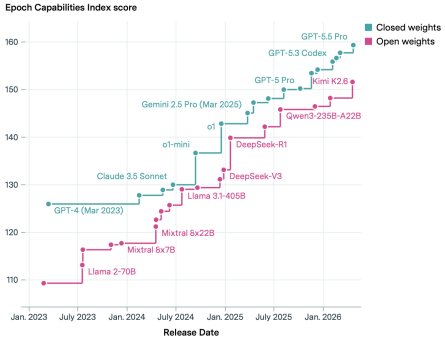
Leveraging Local LLMs

Why local LLMs?

- **Cost**: multi-agent and long-duration autonomy mean huge token consumption
- **Security**: keep sensitive code/data in-house
- **Avoid lock-in / domestic tech**: reduce dependence on commercial services
- **Geopolitics**: today we depend almost entirely on US/China models; cf. the Fable episode (Jun 2026)

Catching up (commercial frontier → local)

- Open models lag the commercial frontier by only 3–4 months (Epoch AI 2026; see figure)
- Consumer GPUs match the frontier of 6–12 months ago (Epoch AI 2025) ⇒ capabilities trickle down to local within months



Epoch Capabilities Index (ECI): closed vs open-weight models.

Source: J. Edwards & L. Emberson, "Open models lag state-of-the-art closed models by 4 months," Epoch AI, 2026 (CC BY). `https://epoch.ai/data-insights/open-closed-eci-gap`

CUDA Porting of Optimized C++: Deoptimize → Reoptimize[6)]

qwen-3-235b-a22b-instruct-2507 (50 trials each; D=Direct, D+R=Deopt-Reopt)

Success rate [%]

Kernel	SS:D	SS:D+R	It:D	It:D+R
Conv2D	12	**96**	58	**90**
BGEMM	0	**94**	14	**96**
DFSpMM	4	**88**	20	**82**
DDGEMM	42	**80**	100	98
GEMM	0	0	2	**84**
SpMM	0	0	28	**88**

Median perf. [GFlops/s, GB/s]

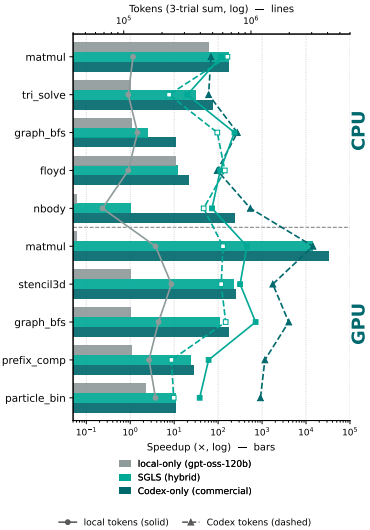
Kernel	SS:D	SS:D+R	It:D	It:D+R
Conv2D	106	**4366**	545	**4389**
BGEMM	–	**354**	1.8	**4474**
DFSpMM	463	**2346**	485	355
DDGEMM	89	**510**	119	**848**
GEMM	–	–	181	**4676**
SpMM	–	–	251	130

- **Background**: when porting highly optimized code to another architecture/language, the original optimizations can hinder portability → simplify via deoptimization
- **Method (C++→CUDA)**: **Two workflows – Direct** (CUDA conversion → reopt.; 2 LLM calls) / **Deopt-Reopt** (deoptimization → CUDA conversion → reopt.; 3 calls). **Two strategies – Single-shot** (each stage once) / **Iterative** (3-parallel × 3 rounds; iterative repair boosts success)
- Effectiveness is problem-dependent, but high when CPU and GPU strategies diverge

[6)] D. Mukunoki et al., "LLM-Based Porting of Optimized C++ to CUDA Through Deoptimization and Reoptimization," GeCoin Workshop in IJCAI-ECAI 2026, Jun. 2026 (accepted).

A Hybrid Approach: Commercial + Local LLMs

- Commercial LLMs are powerful but expensive – the limits of the business model, the shift to API billing (Fable was planned)
- **Role division**: code generation limited to the local LLM (gpt-oss-120b). Commercial Codex (GPT-5.5) writes no code, focusing on **strategy advice**
- Proposed **SGLS** (strategy-guided local search): failure history of iterations decides the search direction
- Representative cases where the hybrid was effective (5 CPU + 5 GPU); SGLS often matches Codex-only in speedup while running codegen locally
- **Tokens** (lines): vs Codex-only, SGLS uses $\sim 2.6\times$ fewer commercial (Codex) tokens by shifting codegen to the local model, at competitive speedup (problem-dependent)



“Information Given” to the LLM vs. “Model Capability”

Background

- Knowing your true performance ceiling (“when to stop”) decides success – and whether the agent’s self-report is trustworthy

Information given

- **none** / **measured table** (FMA peak, STREAM BW – e.g. gcc: 433 GF w/o AVX-512, 1533 GF with the flags) / **design doc**

Findings

- Fable & Opus reach textbook designs with no info (DGEMM nears MKL); the gap is they **don’t measure the ceiling** (Fable stopped >20% short). One measured table fixes calibration (≈86% peak =1326)
- Capability ladder: a read-only table helps even Haiku; a design doc conveys knowledge but not implementation skill → role split – Fable writes the design (“derived doc”), Haiku implements

DGEMM Perf. [GFlops] (n=2048)

Condition	GFlops
Fable 5 · none	1094–1250
Fable 5 · meas. table	1213–**1326**
Opus 4.8 · none	1105–**1209**
Opus 4.8 · meas. table	1074–1143
Haiku 4.5 · none	50–296
Haiku 4.5 · meas. table	48–450
Haiku 4.5 · design doc	253–515
Haiku 4.5 · derived doc	4.3–**1168**
MKL (ref.)	1383
FMA Peak (measured)	1533

Ranges over all trials; bold = best per model. Fair values (meas.-adaptation disabled).

Faster Inference: MoE Expert Selection in Parallel Code Generation

Background

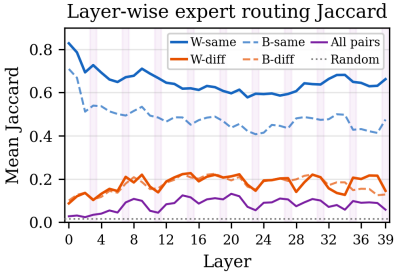
- Local inference on consumer hardware is slow → **faster inference matters**

What we found[a]

- Analyzed expert selection in the MoE (mixture-of-experts) model, across **many parallel code candidates** from one input
- **Roles differ by layer**: input = token-driven, **middle = context-driven** (see figure)
- Shared prefix → **∼2/3 of candidates are essentially identical** (only comments/blank lines differ)

Implications

- Keep middle-layer experts on the GPU + stop duplicate candidates early → efficient inference



Overlap of expert choices between candidates, by layer. Same-position tokens overlap a lot (similar experts); different positions, little. They converge in the middle layers – routing depends on context

[a] S. Hayashi et al., "Layer-wise MoE Routing Locality under Shared-Prefix Code Generation: Token-Identity Decomposition and Compile-Equivalent Fork Redundancy," arXiv:2604.17182, 2026.

Fine-Tuning LLMs for HPC

Training models with code performance in mind[a]

- Trained with GRPO using real-machine code performance (GFlops) as reward. Fine-tuned Qwen2.5 Coder 14B
- Proposed **SQD (Staged Quality-Diversity)**: progressively widening optimization techniques (blocking, SIMD, OpenMP, etc.) while generating/evaluating/training code in a tree-search manner
- Trained on matrix multiply, yet performance improves **even on unseen kernels** (generalization)
- Enables building a performance-optimizing LLM specialized to the target system

[a] R. Mikasa et al., "Improving HPC Code Generation Capability of LLMs via Online Reinforcement Learning with Real-Machine Benchmark Rewards," LLM4HPC 2026 @ ISC26, Jun. 2026.

Evaluation on various kernels (avg.; 5 gens/problem)

Problem	Unit	Before	After
MatMul 1025	GFlops	2.2	**27.4**
MatMul 2048	GFlops	0.5	**24.9**
MatMul (rect.)	GFlops	1.9	**13.4**
DGEMV 8192	GB/s	19.7	**78.3**
SpMM (CSR)	GFlops	5.0	3.8
Jacobi 2D	GB/s	13.6	**22.9**
Prefix Sum	GB/s	0.0	**4.8**

Values are the mean over 5 generations per problem (each code: max of 3 runs after 1 warm-up). Trained only on matrix multiply; other kernels are out-of-distribution

A Benchmark: Precision-Constrained Sparse Solver Generation

(Joint work with Prof. Jézéquel — ongoing)

Task

- Given a sparse $Ax = b$ and a required **true relative residual** τ, generate the **fastest** `solver.c` (C) that meets τ
- Beyond compile / correctness / time, the LLM must **co-design**: matrix diagnosis, solver (CG / MINRES / BiCGSTAB / direct), preconditioner, storage, arithmetic precision, stopping – avoiding over-precision
- Evaluator recomputes the residual in **binary128** $\rightarrow$ independent of the code's self-reported value; only passing solvers are ranked by solve time
- Problems: structure-aware synthetic + SuiteSparse matrices; $\tau \in \{10^{-6}, 10^{-8}, 10^{-10}\}$
- 15 instances, Claude Code: pass rate **Opus 15/15**, Fable 13/15, Sonnet 12/15, Haiku 1/15

Per-instance generation prompt

```
Generate an implementation of solver.c for the
following linear-system instance.

Problem parameters:
- Matrix format: CSR with zero-based col indices
- Shape: {shape}
- Rows: {n}   Columns: {n}   nnz: {nnz}
- Known matrix properties: {symmetry};
    {matrix_desc}
- Required true relative residual: {tau}

Implement only ‘solver.c‘. It must include
‘solver_api.h‘ and define:

int solve_csr(int n, int nnz,
              const int *row_ptr,
              const int *col_idx,
              const double *values,
              const double *b,
              double *x, int max_iter);

The evaluator calls solve_csr with A and b and
checks ||b - A*x||_2 / ||b||_2. Passing
submissions are ranked by measured solve time.

Do not hard-code the solution vector, RHS, output
vector, evaluator outputs, or numerical matrix
values obtained from running the evaluator.
```

Sparse Solver Benchmark: Results (excerpt of 15 instances)

Case	τ	Fable 5	Opus 4.8	Sonnet 5	Haiku 4.5
`tri_well`	10^{-10}	Pass (1.5e-16) 6.5 TD 94.3s, 775k tok	Pass (1.5e-16) 5.4 TD 47.6s, 396k tok	N/G (F) (1.5e-16) 5.7 TD	Pass (1.5e-16) 6.0 TD 45.5s, 253k tok
`anisodiff2d`	10^{-6}	Pass (2.2e-7) 3.8 CG+line-J 161.0s, 1463k tok	Pass (2.2e-7) 3.8 CG+line-J 45.4s, 752k tok	Pass (2.2e-7) 4.0 CG+line-J 98.3s, 1132k tok	N/G (C) (–) – line-ILU
`turon_m`	10^{-10}	Pass (6.7e-11) 1167 scaled MINRES+LD+Q 91.7s, 1234k tok	Pass (6.7e-11) 1173 scaled MINRES+LD+Q 56.3s, 993k tok	Pass (6.7e-11) 1171 scaled MINRES+LD+Q 141.7s, 2687k tok	N/G (T) (–) – scaled Lanczos
`tmt_sym`	10^{-10}	Pass (2.3e-11) 886 scaled CG+LD 110.9s, 1240k tok	Pass (4.0e-11) 1017 scaled CG 64.1s, 822k tok	Pass (4.0e-11) 1026 scaled CG 68.0s, 1017k tok	N/G (C) (–) – CG
`c70`	10^{-6}	Pass (5.5e-7) 421 scaled MINRES 135.1s, 1702k tok	Pass (1.9e-8) 708 scaled MINRES 407.3s, 2872k tok	Pass (5.5e-7) 439 scaled MINRES 59.1s, 1293k tok	N/G (T) (–) – Ruiz MINRES
`raj1`	10^{-6}	Pass (2.2e-7) 297 ILUT+BiCG+LD 240.1s, 2511k tok	Pass (4.9e-7) 369 ILUT+BiCG+LD 39.5s, 500k tok	Pass (4.9e-7) 374 ILUT+BiCG+LD 106.2s, 1154k tok	N/G (T) (–) – Ruiz+ILUT+BiCG
`parabolic_fem`	10^{-10}	Pass (3.0e-11) 894 CG+J 252.1s, 3120k tok	Pass (5.0e-11) 838 CG+J 35.5s, 478k tok	Pass (5.0e-11) 845 CG+J 32.5s, 887k tok	N/G (F) (1.0e-10) 50853 J-CG
`rajat30`	10^{-8}	N/G (G) (–) – gen timeout	Pass (2.5e-9) 7704 Ruiz+ILUT+BiCG+LD 1169.8s, 10689k tok	N/G (G) (–) – gen timeout	N/G (R) (–) – Ruiz+ILUT+BiCG

Each cell: line1 = status, (achieved true relative residual), median solve time [ms]; line2 = strategy; line3 (Pass only) = generation time, total tokens (thousands). Red = fastest passing solve, blue = 2nd, green = fewest tokens among Pass. N/G (F) not accepted, (T) time-out, (C) compile fail, (G) gen-time-out, (R) runtime abort. TD tridiagonal direct, LD long double, Q quad/binary128, scaled/Ruiz equilibration. Full set: Opus 15/15, Fable 13/15, Sonnet 12/15, Haiku 1/15.

Summary

Today's topics

- In just **one year**, the landscape has completely transformed
- Multi-agent has already passed its peak – now a **commonplace technology**
- Development for **specialized hardware**, and code **approaching commercial-library performance**, is now feasible
- **Local technology** has grown in importance (commercial-service dependence, cost, security)
- From automating code development to **automating entire research**

Challenges

- **Accountability and copyright** in code development / autonomous research – the world hasn't caught up
- Will HPC engineers lose their jobs? The **researcher's role also changes** (the bar for research rises – a good thing)
- **Where to run** agents on supercomputers (login / dedicated nodes): we need new operational models assuming resident, long-running jobs

AI's evolution won't stop – how we use it to advance science is what matters