Agentic AI for Software Development in JUNOSW and CEPCSW

Practices and Lessons Learned

Tao Lin (林韬)

IHEP

Quantum Computing and Machine Learning Workshop 2026

2026.08.19

Outline

- Introduction
- Software Development Workflow
- AI Assistance Across the Development Lifecycle
- Some cases
  - CEPCSW Code migration
  - Extending PMTCalibSvc
  - Energy-correction extension
  - Debugging in MC tuning
- Lessons Learned

Introduction

- Hacker and Paoding (黑客与庖丁解牛)
  - Inspired by a talk by Akira Urushibata (漆畑 晶), Beijing, 2014
  - Akira Urushibata, 2018,《論語とコンピュータ》

新聞やテレビではハッカーは悪いことする人ですが、本来は優秀な技術者という意味だったという話は聞かれたことがあると思います。ハッカーの語源は英語の hack という動詞です。斧などで勢いよく叩き割るという意味です。確かに乱暴ですね。しかし、次の漢字を見てみましょう：

分る　解る　判る

どれも字の中に刀が見えます。「わかる」ことは「わける」ことなのです。これだけはありません。「判断」の「断」、「分別」「識別」の「別」、「分析」「解析」の「析」はどれも「分ける」意味の字です。「ハッカー」は理解がすばやい人なのです。

荘子に「庖丁解牛」というエピソードがあります。中国人にハッカーの本来の意味を理解してもらうのにこの話をするといいでしょう。「庖丁」は日本語の「包丁」の語源になった言葉ですが、ここでは人の名前、さらに言うと、「調理場にいる人」という意味です。この人は牛を解体する名人です。彼によれば動物の体には自然にできている隙間があり、そこに刃物を入れれば無理なく切断できる。刃も傷まない。下手な人は固い所を力ずくで折るからすぐ刃が欠けてしまう。ハッカーとは本来、庖丁氏が牛を解体するようにプログラミングを行う人を言う言葉でした。ところで今日は「論語とコンピュータ」という演題ですが、これは老荘思想ですね。

Hack: 劈砍

每个字里都能看到"刀"
"理解"这件事，本质上就是"分开"。
所谓"黑客"，就是理解得很快的人。

黑客一词指的就是像庖丁解牛
那样去编程的人。

Good software development starts with understanding the structure.
Software development is more than writing code.

Software Development Workflow

- JUNOSW/CEPCSW is **a large and complex software system**
  - For example, JUNOSW consists of more than 200 packages
  - Core software: Event Data Model, ROOTIO, Database interface
  - Full processing chain: event generation, detector and electronics simulation, waveform and event reconstruction, and analysis
- Software development follows **an iterative lifecycle**
  - Physics requirements
  - Software and interface design
  - Implementation
  - Build and software testing
  - Code review
  - Physics validation
  - Release
  - Feedback

AI can potentially assist at every stage of this development lifecycle, but its role and the associated risks are different at each stage.

How Development Works in Practice

- Requirements are often incomplete at the beginning.
- New features must be integrated into an existing software architecture.
- Many constraints are implicit in existing interfaces and implementations.
- Knowledge is distributed among software experts, algorithm developers, students, code and documentation.
- A successful change must preserve compatibility and pass both software tests and physics validation.

AI only knows the code and context provided to it; it does not automatically understand the full history and architecture.

AI-Assisted Development

- AI can be used across the development lifecycle.

Stage	AI Assistance
Requirement analysis and design	• Clarify requirements • Identify missing constraints • Propose and compare design options
Implementation and refactoring	• Generate code • Explain unfamiliar APIs • Modify repetitive or well-defined code • Analyze dependencies
Testing and validation	• Generate test cases • Analyze build and CI failures • Compare outputs and summarize differences
Debugging	• Analyze logs • Propose possible causes • Construct minimal reproducing cases

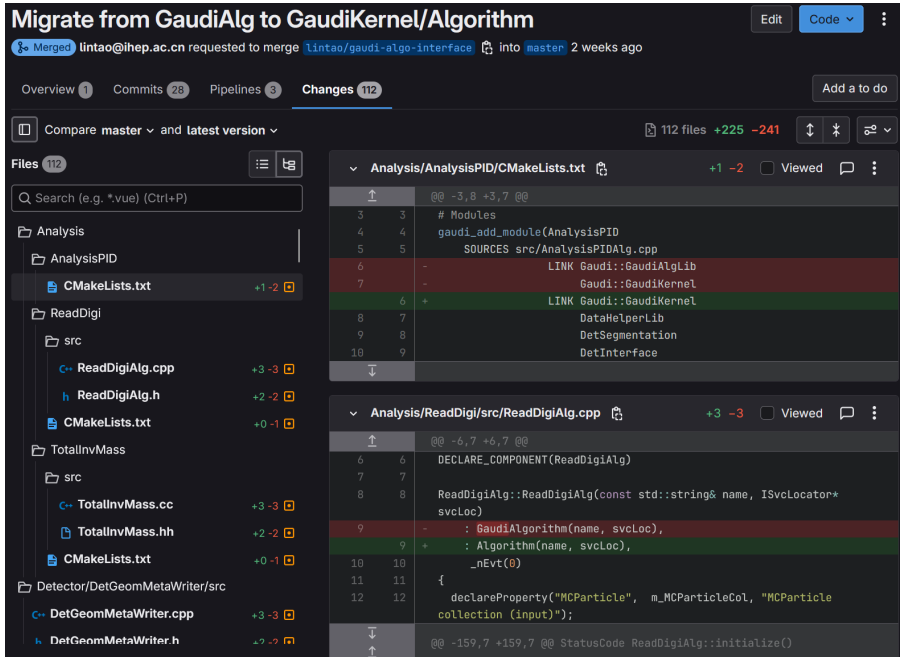
Several Cases

Case	AI usage mode	Outcome
CEPCSW Code migration	Agentic AI	Successfully completed
PMTCalibSvc extension	Expert-led collaboration	Successfully completed
Energy-correction extension	Feature implementation delegated to AI	Completed after collaboration with expert
Debugging in MC tuning	Agentic investigation	Partially localized; unresolved

These cases represent different levels of AI autonomy and different levels of expert judgment required.

Case 0: CEPCSW Code Migration

- Motivation and Requirement
  - The TDR baseline is frozen on LCG 105 to ensure reproducible production.
  - Goal: Need to migrate the CEPCSW to LCG 109
- AI usage mode: Agentic AI workflow
  - Identify affected packages
  - Modify packages one by one
  - Build and debug iteratively
- Result: migration completed
- Lesson: the agent can search the codebase, apply systematic changes, build the software, and iterate over errors.



112 files modified. 225 lines changed.

Case 1: Extending PMTCalibSvc

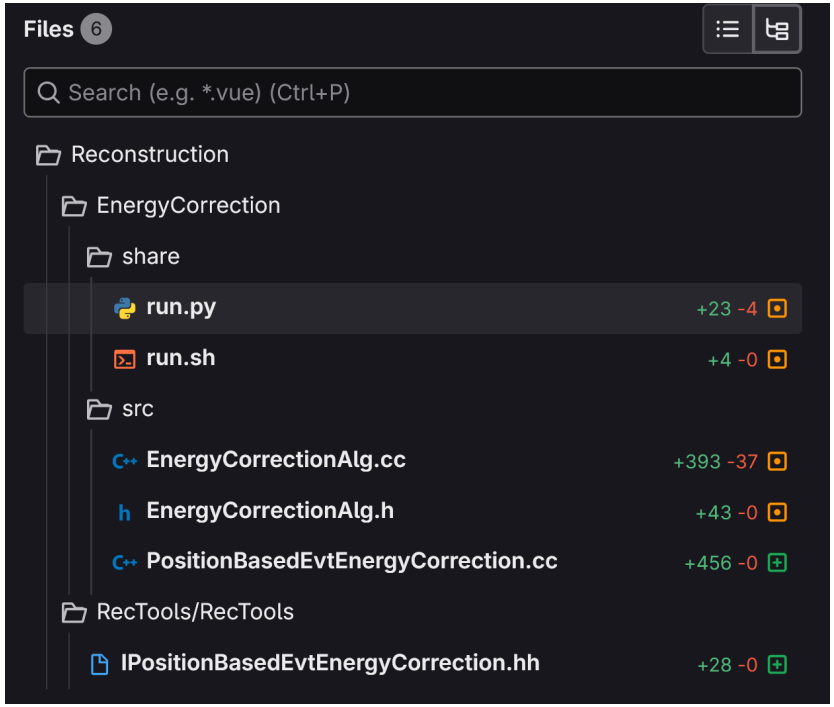
- Motivation and Requirement
  - Originally, PMTCalibSvc could provide calibration constants from either COTI or Deconvolution, but not both simultaneously.
  - The default configuration used the COTI constants, while the Deconvolution algorithm required its own set.
  - Goal: expose both sets through the same service while maintaining backward compatibility.
- AI usage mode: expert-led collaboration
  - Discussed several design alternatives with AI
  - Chose the design based on my JUNOSW experience and compatibility requirement.
  - Divided the implementation into small, reviewable steps
  - Reviewed, compiled, and tested each step
- Result: The extension was completed without breaking the existing interface.
- Lesson: AI explored the design options; the developer retained ownership of the architectural decision.

Case 2: Energy-correction extension

- Motivation and Requirement
  - Originally, the energy correction was time-based:
    - `CorrectEventEnergy(double originalE, const TTimeStamp& t)`
  - New requirement: add a position-dependent correction

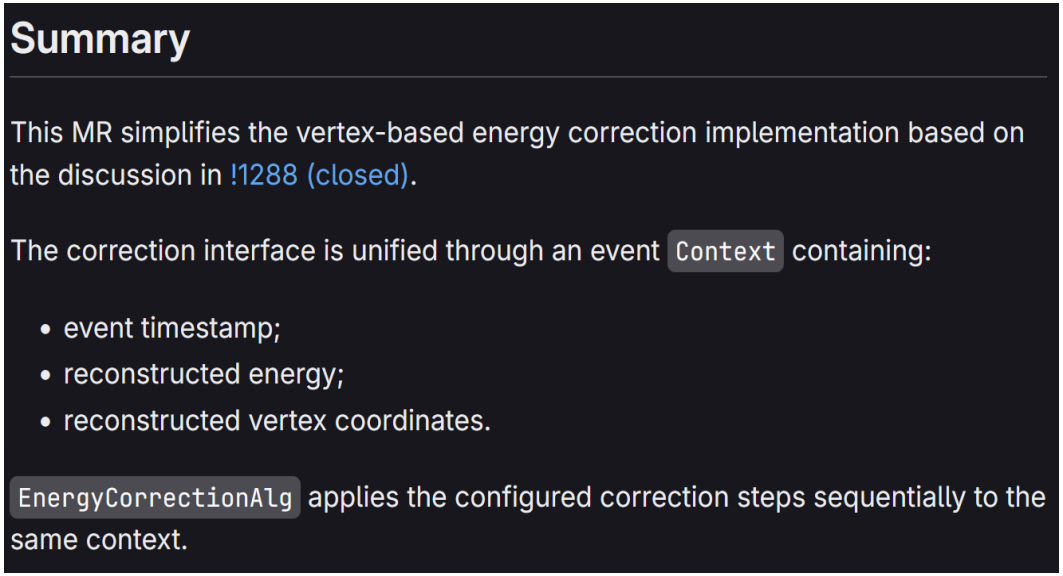
- First attempt by Shubing (MR !1288)
  - AI was asked mainly to implement the new functionality.
  - AI produced a large patch that bypassed the existing interface
  - EnergyCorrectionAlg was almost completely rewritten.

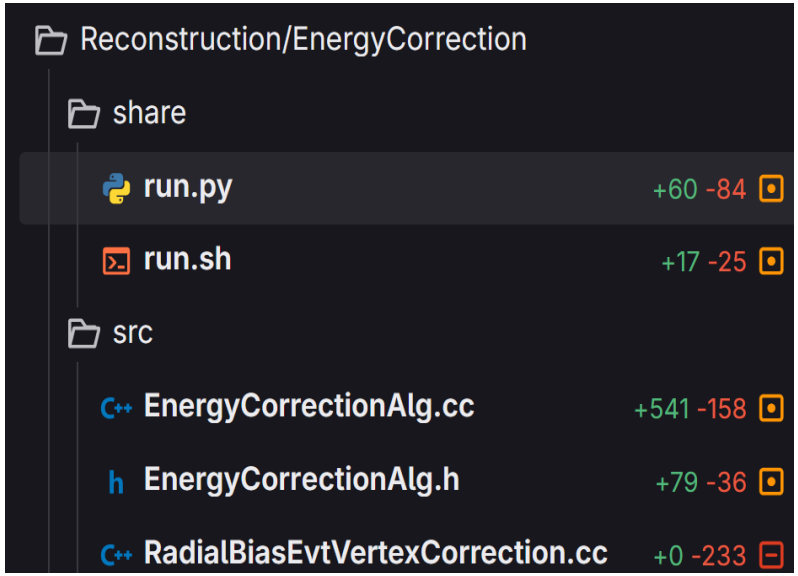
- Lesson: The feature was specified, but the architectural constraints were not



Case 2: Energy-correction extension

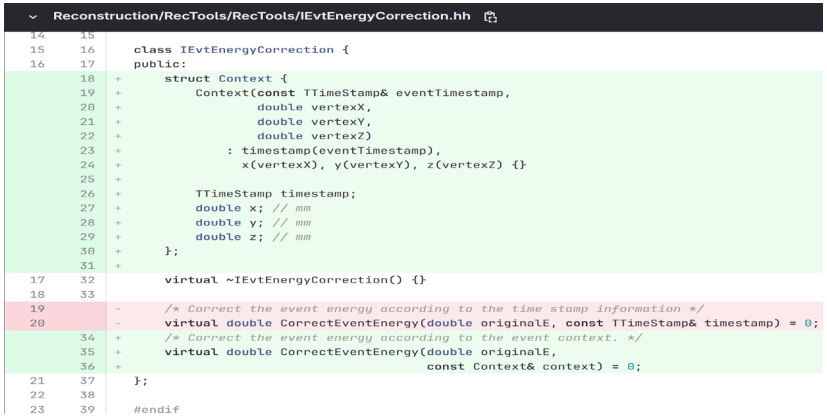
- After reviewing the MR, I had a short discussion with Shubing
  - I suggested introducing a Context to avoid adding another interfaces.
- Second attempt by Shubing (MR !1302)
  - AI removed the original interface and rewrote the algorithm.
- Lesson: A design hint without explicit compatibility constraints can trigger an overly broad rewrite.

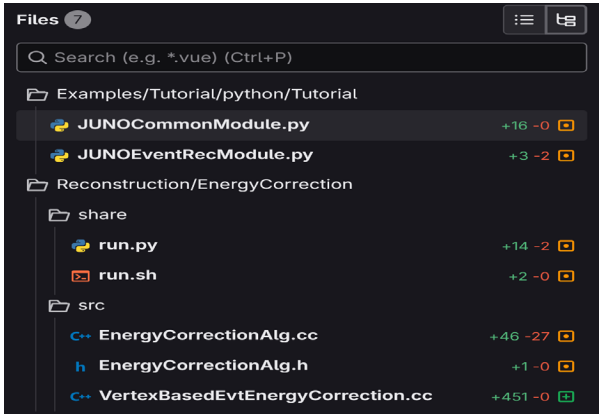




Case 2: Energy-Correction Extension

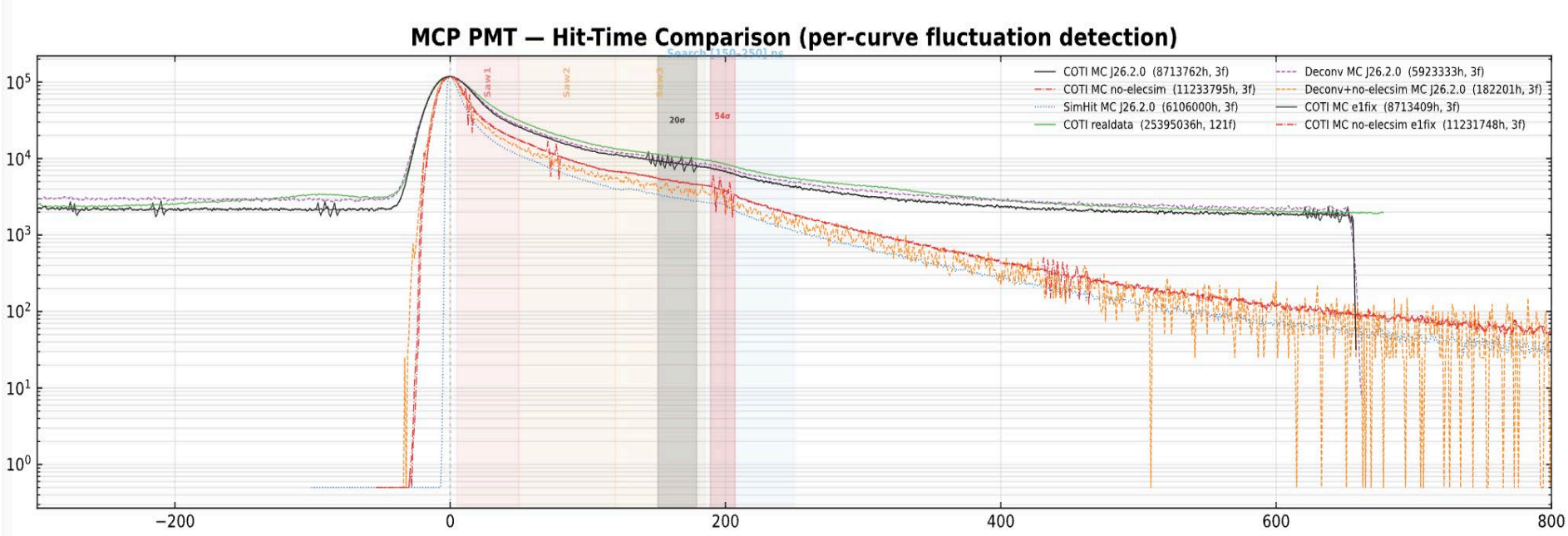
- Third attempt: Recovery with a staged plan
  - I had a dedicated meeting with Shubing to clarify the architecture and compatibility constraints.
  - We divided the implementation into two steps:
  - Step 1: Extend the existing interface with a Context carrying the required event information.
  - Step 2: implement a new correction tool using the extended interface.
- Result: The extension was completed through small, reviewable changes.
- Lesson: Separate interface evolution from feature implementation.





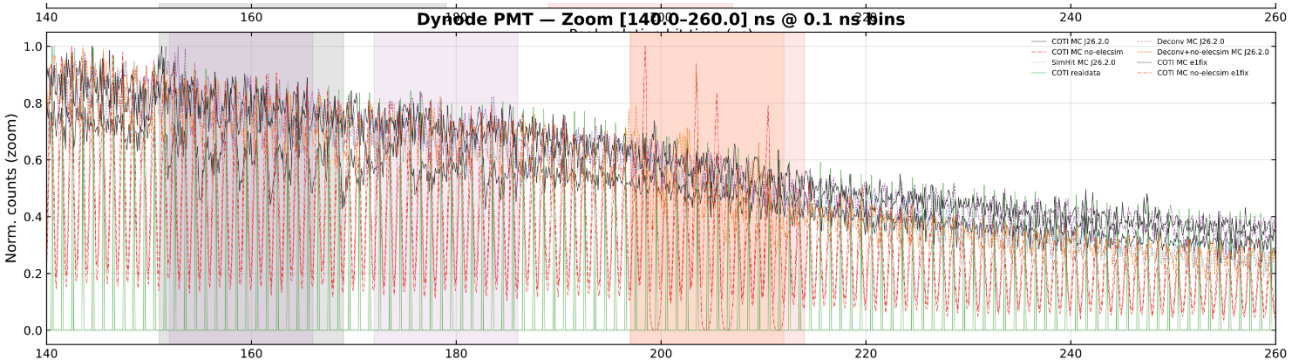
Case 3: Agentic AI for debugging in MC tuning

- Motivation and Requirement
  - Simulated hit-time distributions showed anomalous oscillations that were absent in real data.
  - Similar structures appeared with both COTI and Deconvolution, although their amplitudes differed.
  - Goal: Identify the root cause of the issue and fix the fluctuations in the simulated-data reconstruction.



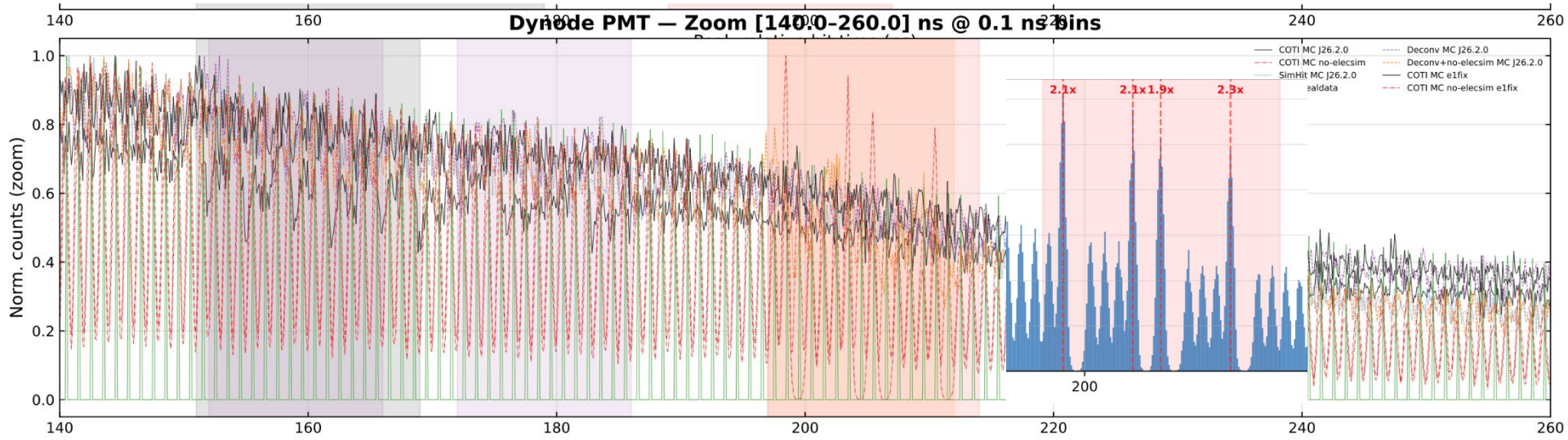
Case 3: Agentic AI for debugging in MC tuning

- AI usage mode: **agentic exploration with limited expert guidance**
  - The AI explores autonomously, generates hypotheses, and returns results for evaluation by non-expert students.
  - The DeepSeek V4 Pro non-multimodal version is used; thus**, a human is required to interpret visual content**.
- First Attempt by Zhihao Li (non-expert student)
  - The AI consistently believed that the issue lay in the **reconstruction** stage.
  - From plots, we saw both algorithms fluctuate (COTI stronger, deconvolution weaker) and redirected AI from ***reconstruction*** to the ***simulation*** *domain*.
  - **Lesson:** AI misdiagnosed due to **lack of visual interpretation**; the complex, distinct fluctuation patterns of COTI and deconv are obvious to humans but hard to discern via code alone.



Case 3: Agentic AI for debugging in MC tuning

- Second Attempt with expert guidance
  - Guided by experts Ziyan Deng, Tao Lin, Peidong Yu – disabled electronics effects one by one to isolate the cause.
  - After removing all electronics effects, fluctuations persisted – root cause traced to the simulation domain beyond electronics.
  - Plot reading revealed the specific pattern: adjacent wave peaks and troughs merge into one.



Case 3: Agentic AI for debugging in MC tuning

- Result:
  - The issue was localized to the electronics simulation stage.
  - However, the exact cause and a validated fix remain unknown.
- Lesson:
  - Domain expertise is essential for designing discriminating tests and interpreting ambiguous evidence.
  - Agentic AI requires access to all relevant evidence—including plots—and a reliable validation loop.
  - A stronger or multimodal model may help, but model capability alone cannot replace controlled experiments and validation.

Lessons Learned (1)

- Keep architectural decisions under human ownership.
  - Use AI to explore design options, not to outsource technical judgment.
- Make the context and constraints explicit.
  - Specify existing interfaces, compatibility requirements, scope, and expected behavior.
- Separate interface evolution from feature implementation.
  - Avoid mixing architectural changes with the implementation of new functionality.
- Work incrementally with clear checkpoints.
  - Keep changes small and reviewable; compile, test, and validate after each step.
- Build a reliable feedback loop for agentic AI.
  - Successful automation requires observable evidence and clear success or failure criteria.
- **AI-assisted development works best as a controlled collaboration guided by domain expertise and continuous validation.**

Lessons Learned (2)

- AI compresses the development cycle
  - AI gives us a much sharper knife.
  - Search, implement, build, test, compare, summary become much faster.
- Fundamentals remain
  - We still need to know where to cut.
  - Architecture, decomposition, constraints, validation criteria, physics judgement, final decision
- Human judgment becomes more important, not less
  - More decisions arrive in a shorter amount of time.
  - Human attention and judgment become the bottleneck

Good judgment requires strong fundamentals.
Know where to cut.
Know where to spend human attention.

Thank you