AI for Research Software Engineering

Jeremy L Thompson

Helmholtz-Zentrum Hereon

jeremy@jeremylt.org

14 July 2026

Overview

Research Software Engineers must respect both
software development best practices and **research ethics**

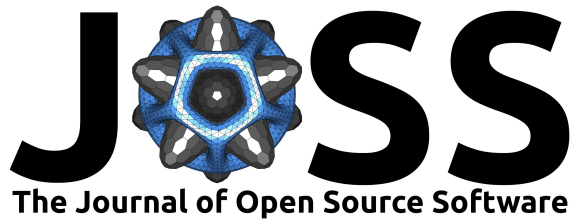
Not all development best practices for LLM (AI) based tooling
map easily from industry to research software development

Research ethics impose additional constraints beyond
industry software development considerations

Key takeaway: **LLMs are not a magic bullet**

Software as First Class Artifacts

Many academic groups (San Francisco DORA, JOSS, etc) advocate for research software as first class artifacts (like papers)



https://sfdora.org

https://joss.theoj.org

RSEs need to ensure **quality**, **clarity**, **testing**, and **documentation** to deliver first class software

Overview

Industry Best Practices

LLM based coding tools are subject to GIGO

Garbage In $\rightarrow$ Garbage Out

Best practices for input (codebases, planning) and output (process, review)

'Healthier' codebases more likely to result in higher quality LLM output

Best practices for human development $\rightarrow$ good codebases for LLM

High developer skill required for higher quality LLM output

Caveats

Quality research on LLM best practices hard to identify

- Massive amounts of money in LLM research
- Profit motives can complicate research context
- Modern research more likely to include positive, definitive wording (LLM are prone to this as well)
- Included impressions from professional discussions, blogs, etc

Software Development Life Cycle

SDLC (Software Development Life Cycle) roughly has 3 parts

- Planning: what should be coded and how?
- Development: write the code!
- Review: did we write the right code?

Industry best practices require SDLC while academic coding often doesn't

LLM code generation moves more effort into Planning & Review

Software Development Life Cycle

Good SDLC (Software Development Life Cycle) helps guide LLM usage

- Planning: RSEs decide what code is needed
- Development: LLM prompted to generate code
- Review: RSEs decide if the generated code is correct

Planning & Review critical to ensure higher quality LLM output
(See also Test Driven Design (TDD))

This same pattern can be included in smaller loops to control risk/error

Code Health

CodeHealth™ is a quality metric used by CodeScene (paid software suite)

Corresponds to many 'code smell' concerns

- Code cohesion, complexity
- Adherence to DRY
- Object and method complexity

Higher CodeHealth $\rightarrow$ fewer failing tests after LLM refactoring

Personal Discussions

Personal discussions and blogs largely align with CodeScene research

Skilled developers needed to plan, monitor, fix LLM output quality

A clear plan and 'definition of done' before development helps greatly

Overall design and style consistency of particular note

High developer skill required for higher quality LLM output

High Skill Needed

RSE organizations frequently list skills and training as a weakness

- Lack of recognition of research software development as *scientific* contribution;
- Missing link between research software development and the scientific reputation system;
- Limited availability and usability of research software;
- Redundant and parallel software developments;
- Insufficient skills in software engineering;
- Lack of quality standards for the development and review of research software;
- Lack of reproducibility;
- Unclear rules for publication with regard to licenses and intellectual property (IP).

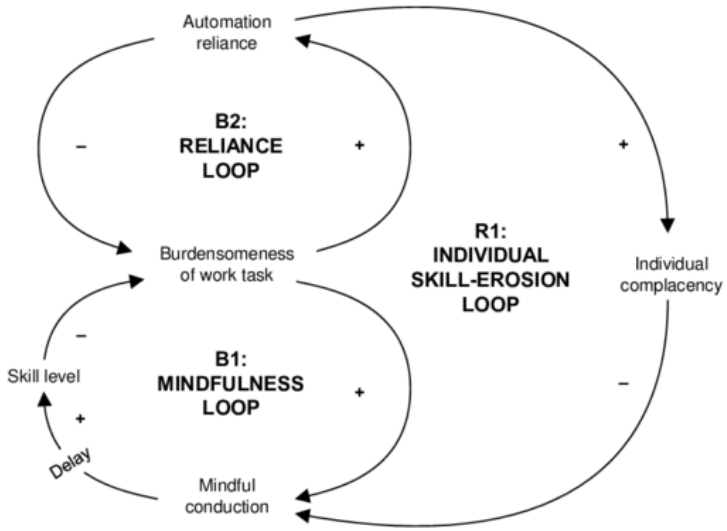
deRSE list of known community issues

LLM is not a replacement for professional software skills

Unfortunately, non-careful LLM usage can also inhibit skill formation

De-skilling

LLM present risk of de-skilling when too much cognition is offloaded



Skill erosion loop (Rinta-Kahila et al.)

Research software as first class artifacts requires clarity, correctness

De-skilling presents an important risk to clarity and correctness for RSE

Summary

RSEs need **training in development best practices** to best use LLM

Skills and training is a **known weakness** for RSEs

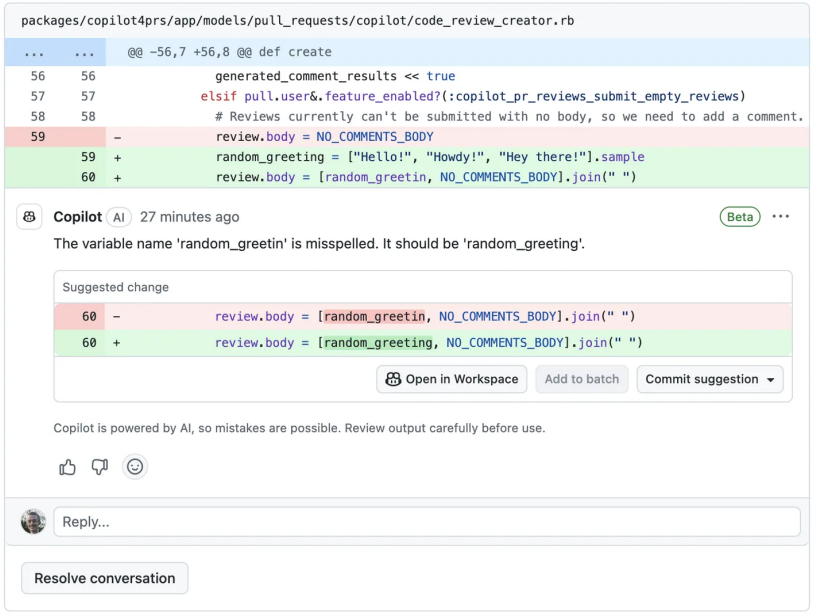
LLM usage can result in **de-skilling** or **inhibit skill formation**

Curating research software like papers parallels industry best practices

Recommendation: Ensure good development skills before using LLM

Code Review

LLM based code review is a promising application



Code review is high skill activity, so de-skilling risk still applies

Note: LLM based code review appears to consume many tokens

Options

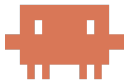
Some popular LLM based code review tools:

- GitHub Copilot
- Coderabbit
- Cursor
- Claude Code
- And many, many more

Caveats

Total token usage and costs per token continue to rise

Most popular models controlled by large US tech companies



SLMs (Small Language Models) on local hardware possible alternative

SLM options: Qwen, Gemma, Phi, SmolLM, Ministral, etc

Offers data control, no API costs, but less context and complexity

Relevant Points

Three key ethics points:

- Research requires **transparency** in methods, including tools
- Software must remain **legitimate and citable** research
- Must **not plagiarize** work of others, even inadvertently

See European Code of Conduct for Research Integrity, among others

Disclosure

Disclosure is a natural requirement

- Disclosure is at the heart of science
- Disclosure improves reproducibility
- Successful methods can be re-used
- Unsuccessful methods can be avoided

BLUF: If you are uncomfortable stating what you did
then the software/research is not ready to release and share

Open Source Policies

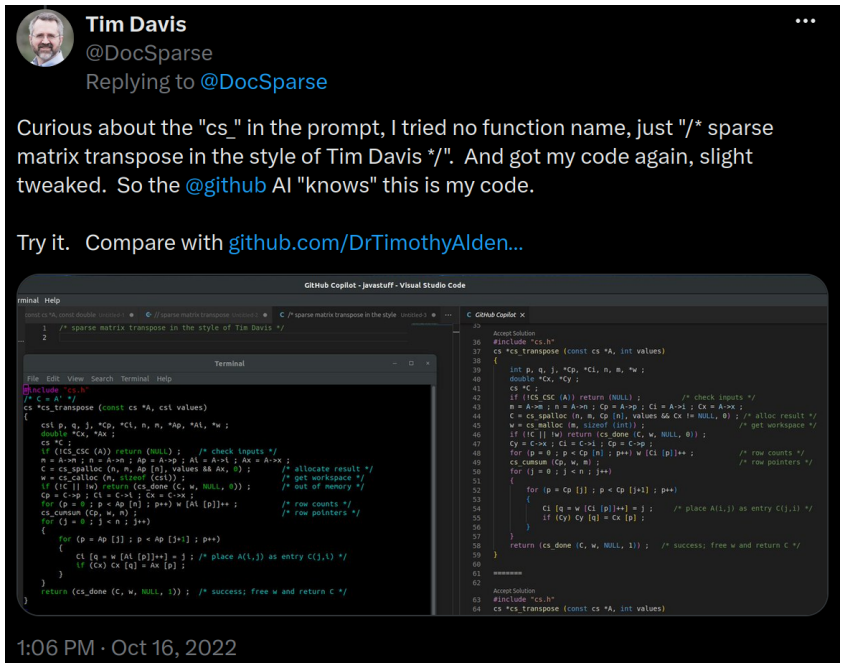
Disclosure aligns with Open Source policies permitting LLM

- Many open source project have LLM policies
- Disclosure usually required if LLM output permitted
- Disclosure can improve code review

See also Melissa Weber Mendonça's list of open source policies:
https://github.com/melissawm/open-source-ai-contribution-policies

Plagiarism

Dr Tim Davis prompted Copilot to generate his copyrighted CSparse code
LLM generated code without attribution, violating LGPL licensing



Researchers are ultimately responsible for the code they create

Accidental plagiarism via LLM is still an ethics violation

Citation Errors

GPT-4 has fabricated or generated erroneous citations

Generated citations can have slightly flawed data

LLM output can omit specific training data related to the prompt

RSE should guard against research process inversion:
research $\rightarrow$ reach conclusions ✓
conclusions $\rightarrow$ find research to support ×

Citation Bias

LLM generated citations reinforce bias in citations

Summary

Easy to accidentally plagiarize with heavy usage

LLM generated citations can be **fabricated, flawed, or reinforce bias**

RSEs are ultimately responsible for our work, no matter the process

Summary

Key takeaway: **LLMs are not a magic bullet**

RSEs who let LLM code for them will have bad code

We need to ensure good development skills before integrating LLM

RSEs who let LLM research for them will violate ethics

We are ultimately responsible for our work, no matter the process

Questions? Comments?

Community comments, resources, and questions invited

https://github.com/jeremylt/personal_website/pull/3

Key takeaway: **LLMs are not a magic bullet**

RSE need good development skills before integrating LLM

RSEs are ultimately responsible for our work, no matter the process

AI for Research Software Engineering

Jeremy L Thompson

Helmholtz-Zentrum Hereon

jeremy@jeremylt.org

14 July 2026