HPC-BP Webinar Q&A

Using Generative AI for Coding Tasks in Scientific Software

Date: 9 July 2025

Presented by: Anshu Dubey (Argonne National Laboratory), and Akash Dhruv (Argonne National Laboratory)

(The slides are available via the link in the page's sidebar.)

Q: How and how much are you training the computer to understand how you code so it can assist you? Did you submit prior fortran code samples from your past?

Anshu: No I did not. I am not training it at all.

Q: Are you using a commercial AI Coding Assistant (Azure, OpenAI, or Oracle, etc.) or does Argonne have a GPT like coding assistant?

Anshu: Argonne has its own, but I'm using a commercial one, OpenAI O4 mini.

Q:How many attempts did you make at creating the final (rather complex!) prompts that you show on the screen?

Anshu: I had 4 rounds of iteration. These prompts (Slide 18) were at most 2-3 iterations each and these (slide 23) took a lot of work. Wasn't that I changed the prompts, but needed to work through the problem with paper and pen to design.

Q: Before you write the prompts, you have to have a clear idea of how you organize the algorithm–maybe you have a flow chart written down. You are just asking LLM to do the busy work. Is it fair to say that your prompts are your codes? And LLM is a very high level language?

Anshu: Exactly my perspective. What LLMs have done for me has freed me from the tyranny of syntax.

Q: If you end up keeping some of the code for Flash-X, are there any caveats with code attribution and licensing? Does it impact your Apache-2 licensing?

Anshu: About licensing I would hope not, but it is a bigger issue for the whole open source community to consider and grapple with. I think what is going to happen is we will acknowledge the code generated with LLMs as such, but also have it rigorously verified; nothing is accepted unless it passes the tests we have.

Q: We didn't really get an answer about licensing. There are a lot of concerns about intellectual property and LLMs. How can we say for sure that it is ok to include LLM-generated code in something with specific licensing like Flash-X which may be incompatible with the original licensing of the codes used to train the LLM.

A: That is an issue that licensing authorities will have to grapple with. Flash-X has an open source Apache2.0 license, which is about as flexible as it gets. If we run into trouble with generated code, then the whole enterprise of using code generation with LLM is a problem for all of science.

Moderator's note: IP issues around AI is still a very dynamic question and I don't think anyone can state a definitive answer at this time. I think current best practice would be to make sure you know where you're using AI-generated code in case it might need to be replaced in the future, once we have more definitive guidance on the legal issues.

Q: Do you have any concerns about giving your intellectual property away during prompting or are these open source systems where you already have provided your code online?

A: The code that we used for prompting has an open-source license.

Q: What do you consider your source code? The prompts? The generated source?

Anshu: At the moment it is both. But more and more we are working with converting chunks of code into recipes, and performance portability layer. Whichever part of the code we work on with our LLMs and performance portability layer we will not have any Fortran in it. We envision a situation where we have some script based code generation engines, some LLM based code generation engines, so that we can generate code in whichever language we want on the fly. We would target C or F90. But that is my pie in the sky vision. I don't know how realizable it is.

Q:Did you use the LLM to generate any automatically running tests?

A: No. I couldn't think of a way that I would have confidence in any LLM generated automatically running code.

Q: How would you modify the methodology if you needed the new code to use existing types and procedures/methods? I have found LLM very useful for prototyping but not so much for the subsequent integration into an existing code base unless the functionality is completely orthogonal.

A: The code I am developing will go and fit into existing code which already has its own defined data structure for particles and methods for moving the particles etc. What I have done is to make sure that the data structures I am asking it to generate are identical to the ones existing in the code. And the fake utilities I am using have identical interfaces to the ones that exist in the code. That said, it will not eliminate all manual work in integrating. Also, I have had almost no success in refactoring existing code so far. Haven't figured out how to phrase my prompts in ways that are faster than doing it manually.

Reply: Yes - that last bit is a very important aspect to communicate. Am crossing fingers that better refactoring emerges in this context.

Q: Thanks for the presentation! Anshu mentioned that experimenting with four different approaches was helpful, discarding old approaches. How was this workflow achieved? E.g. git commit first approach in branch. Toss code and commit next impl. etc.

A: That is pretty much how it is. But I am not even committing every version. So far what has happened is that I think of alternative ways before I am ready to declare it even a prototype.

Q: Have you tried optimizing the code for HPC architectures using specific libraries?

Anshu: No, not using AI tools.

Q: What is the reproducibility of the code using the same prompt? What if you use a different LLM model?

A: It would be different code. Though I have succeeded in making it more or less deterministic if I ask it to include arithmetic exactly as I describe it.

Related discussion from chat:

> J: It probably can't be assumed the LLM will emit code deterministically right? As in over time the prompts may generate different code?
>
> A**:** That is true and it happened to me. But if your prompt is simple enough you can get roughly similar code. But because you get different generated code is why for now at least we will keep the generated code in the repository.
> And maybe that will never change. We will just have to wait and see.
>
> B1: True Jon. And if prompts are part of the source code, I wonder if they should be stored somewhere close to the code that they generate.
>
> B2**:** The Model Context Protocol docs on prompts could be a useful resource for this. https://modelcontextprotocol.io/docs/concepts/prompts
>
> A**:** They are stored in the code module where they belong as txt files.

Q: I do not see how the prompts can ever be a replacement for actual source code. LLMs can give different results each time and may or may not break some tests. And more practically, code generation can take some time in my experience and thus impact build times.

A: Like I said, that may or may not happen. Right now we keep both the prompt and source code, and treat prompts more like documentation. But I have also experienced that if your prompt is pretty much an implementation in English you get the same code. Which means the arithmetic you want to do becomes a part of your prompt.

Q: Can you reverse engineer the English prompt “code” from Fortran and then use it to generate code in a different language? Can you similarly take existing code, reverse engineer to English and modify the English to generate similar code?

A: I can use the same prompt to get code in C. I have tried to do reverse engineering from Fortran for code refactoring, but haven't had any success with that at all. But of

course it could be because I am not able to figure out the right kind of prompt to make that happen.

Q: Have you ever tried comparing performances of different LLMs?

Akash: We have compared performance in terms of accuracy in our paper. However performance in terms of speed is difficult because we use high parameter count models through their APIs and lower models locally. It would be good to compare all models on a single system for that.

Q: Was your use of the coding assistant for the Argonne Aurora Exascale System? I'm desperate to learn best coding practices for this new system.

Akash: Currently we are not using Aurora for any of this work.

Anshu: We have not tried that; other efforts within Argonne trying to do this we can find out more about.

Moderator: David: For best practices using Aurora separate from AI, ALCF runs an extensive training program. See https://alcf.anl.gov/events. Note that many of their past trainings have been recorded. Also, note that many other computing facilities offering extensive training programs too, though obviously they might be targeting different hardware.

Q: Why did you decide to apply this coding assistant to Fortran?

Anshu: Because our code is in Fortran

Q: How does the LLM context window limit you to very short files? Is it because the context window expects natural language?

A: Yes. All the contents of the file, along with the chat completion template are processed as natural language.

Q: Using LLM to write code is supposed to save time and effort. But after LLM gives us a code, it may or may not be right. We still need to go back to fix things. Sometimes

learning somebody else's code (in this case LLM's code) is almost as much work as writing the code ourselves. In your experience, is LLM really saving you time and effort?

Anshu: For me, definitely. Whenever I'm writing code I inadvertently introduce bugs that take time to clean up. LLM code typically does not have typographical bugs. When it doesn't take me an hour writing a different version of the test I'm more willing to do it. Third, if you're asking the LLM to do routine functions it has seen before, like some of the utility functions for Flash-X, it gets them right the first time. I created a separate test harness.

Akash: This was a new code for me. AI helped me understand the source code better by solving LLM hallucination and bugs. AI was helpful to learn the code itself. Anshu: It turned it into small enough files to digest. Akash: Yes, then the task is in the small files.

Q: There has existed a utility, f2c, for converting FORTRAN to C (and that was later expanded to generate C++, I think). Why not use a tool like that to convert your FORTRAN to C/C++? Did you compare your LLM generated code with C/C++ generated by such a legacy tool? (See also: https://en.wikipedia.org/wiki/F2c and https://manpages.ubuntu.com/manpages/jammy/man1/f2c.1.html)

Akash: I haven't compared f2c with codescribe yet. The challenge with f2c was the directory structure. Would be good to do a more robust analysis.
Anshu note: one thing to figure out would be how well does it handle "use" statements when the used module exists elsewhere in the code.

Q: The code translation problem seems to me that it could be handled by the technology in compilers (i.e. code rewrite rules). From a principled/theoretical point of view, would it not make more sense to implement this capability as a compiler, rather than an LLM? Further, the way the languages are defined there is not necessarily a perfect translation from one language to another. It doesn't seem to me that this approach would address that problem, do you agree?

Anshu: I would agree that it isn't possible to do exact code translation in all situations, at least for now human intervention comes in in those situations. As far as using compiler technology is concerned, I would have imagined that LLVM and MLIR would have made it possible for all languages to use common infrastructure below IR level, but people doing compilers tell me that that hasn't happened, so I am not at all clear about how

well language translation will work based on compiler technologies. If someone has the incentive to do it, it would definitely be more reliable than doing with LLMs. At the moment though, the point is LLMs are available, compiler based technologies are not. Akash: This field is rapidly evolving; there may be IDEs using LLMs helping translate your code base. Anshu: In the scientific world, the cottage industry of translating code is compatible with Kokkos. Not sure why there isn't a solution; may not be an easy solution to implement.

Related comment from the audience:

> Banks have built a lot of code translation. I'm studying at University of Cambridge (UK) and they have OSS conversion partnerships with a lot of firms to convert code. Just fyi

Q: For giving the LLM extra information about the existing functionality of the source code, are you using something like Language Server Protocol https://langserver.org/ or a custom solution?

A: We have our custom solution by creating a RAG database, using Language Server Protocol is a better idea definitely and I am exploring it to make it part of CodeScribe.

Q: Can we get a live demo?

Akash: I am preparing some tutorials that I will put in the github repo for codescribe: github.com/akashdhruv/CodeScribe

Q: Rather than use LLMs, which do not have reproducible output, why not use existing code generation technology? Domain specific languages (DSLs) exist, and can be created, that act like a higher level language and generate code, the same model as being proposed for LLMs, but the DSLs are reproducible. Scientific software is well understood, so it is possible to capture domain specific knowledge in a DSL. This also removes the concerns about licensing.

Anshu: I don't know of a general purpose DSL that can do arbitrary generation like LLMs can. They were gaining ground 10 or so years ago, but most of the groups that I know of that developed DSLs at that time have abandoned them because corner cases arise, and maintenance becomes very expensive. A handful have been successful, but for the most part by the time you get DSLs up to the point where it can handle all outliers for code generation it begins to look like a regular language with as many

features and your code becomes as complex as the code written in other programming languages. Does not free you from the tyranny of syntax like LLMs do. With robust verification using an LLM is a viable option now. For every DSL someone has to write a compiler and keep it up-to-date.

Moderator: Generating Fortran OpenMP tests with ChatGPT; one of the most common problems was ChatGPT declaring variables at places in the code not allowed in Fortran (whereas they would be allowed in C), so LLMs can also suffer from the tyranny of syntax!

Akash: DSL is more relevant for generating new code.

Moderator: Anshu's use case was exploring a new algorithm to my thinking is a step before designing a DSL.

Q: Why is GPT4 working better than the other AI for translating code from Fortran to C++?

Akash: GPT 4 has high parameter count which is a result of more rigorous training. I believe if you were training Codellama on specific instruction-set for code-translation it would perform well.

Q: How does codescribe work when the individual files are large that they exceed the context window ?

Akash: This is a challenge. I have found success in breaking up the file to reduce context size.