AI-Driven Development: Accelerating Code and Proposal Writing with Next-Gen Tools

May 30, 2025

Chunhua “Leo” Liao

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Lawrence Livermore National Laboratory

Agenda

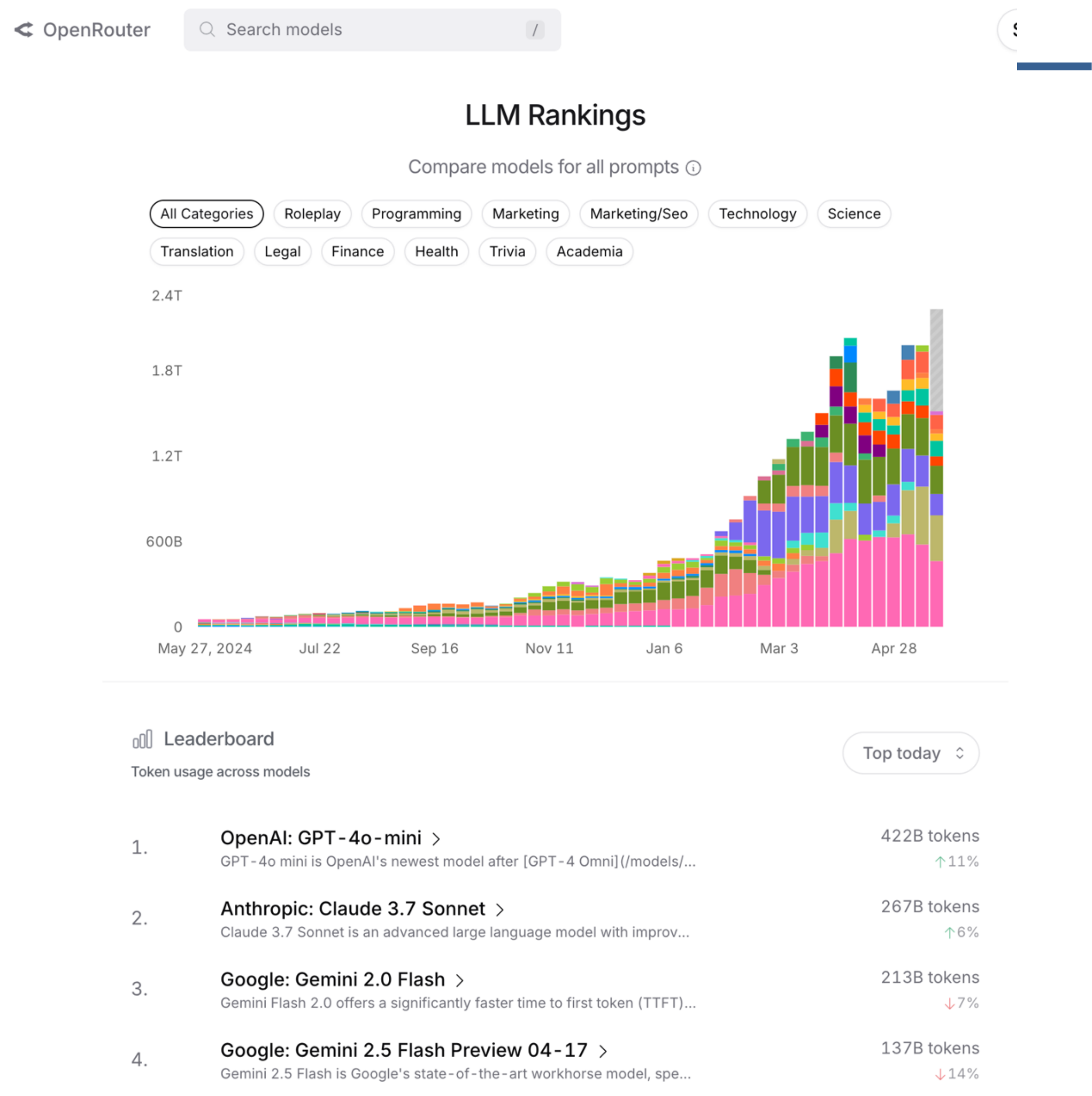
- Introduction to AI tools
- Three research tools under development
  - Fortran2Cpp
  - CompilerGPT
  - Code2Doc
- A few other AI tools (demos)
  - DeepWiki
  - VS Code + Cline + LLMs
    - Help reproduce a paper
    - Help proposal writing
  - alphaXiv (if time permits)
- Conclusion

What is Generative AI?

- **Definition:** Generative AI refers to AI systems (often deep neural networks) that create new content (text, images, music, etc.) based on patterns learned from training data.
  - **Examples:** Large Language Models (LLMs) like OpenAI's ChatGPT and Anthropic's Claude
- **How it works (high-level):** These models are trained on massive text datasets to predict the next word in a sentence.
  - GPT, for instance, stands for **"Generative Pre-trained Transformer,"** indicating it uses the Transformer neural network architecture pre-trained on enormous text corpora. This architecture allows the model to understand context and produce human-like language.
- **Key point for beginners:** Generative AIs don't "think" like humans – they statistically generate likely sequences of words.
  - They produce impressively human-like results but can also make mistakes or nonsensical statements if prompted oddly. Always remember it's pattern-based generation, not a perfect oracle of truth.
- **Recent update - reasoning AI**: go further by evaluating, planning, and making logical inferences - mimicking how humans think through problems.

Two Ways to Interact with AI Models

- Web Interface
  - https://claude.ai/ or https://chatgpt.com/ for example
  - Type in questions or requests in natural languages
  - Advanced features: choose models, upload files, search, deep research, voice mode …
- Programming Interface: API
  - Writing programs that send requests and get results via API
  - OpenRouter: one API for all models



More Specialized AI Tools for Software Engineers and Scientists

Coding	Research and Writing
• GitHub Copilot, OpenAI Codex, Claude Code, Google Jules • VS Code + Cline extension • Cursor, Windsurf • Devin, OpenHands • Algorithm discovery and optimization: AlphaEvolve • Documentation generation: DeepWiki	• Reading papers: alphaXiv • Literature Review: OpenAI Deep Research, similar feature from Grok, Gemini, and Perplexity • Hypothesis Generation: Google AI Co-Scientist • Error checking: YesNoError • Fully-Automated: AI Scientist, AI Researcher, Agent Laboratory

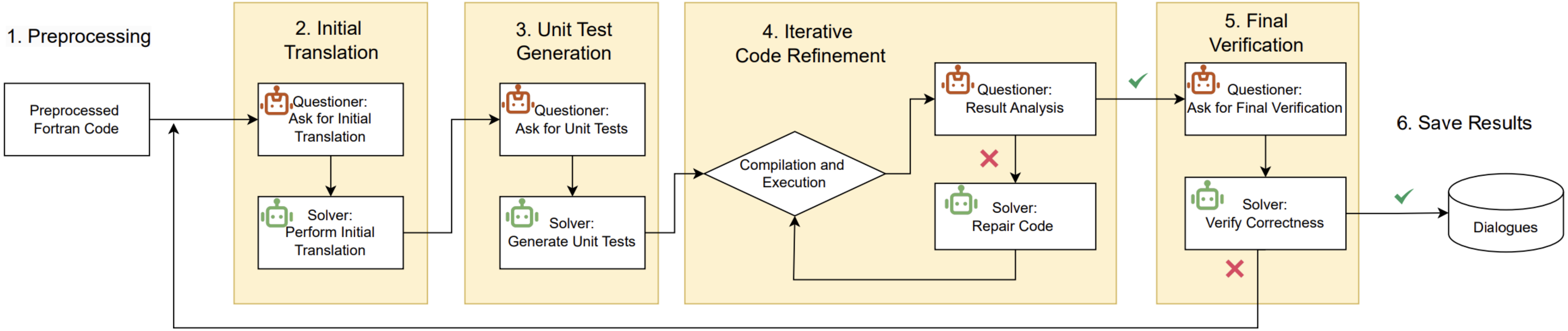
A few AI-based Tools We are Working on*

- Fortran2Cpp: Migrating Fortran to C++
- CompilerGPT: Making compiler diagnostics interpretable and actionable
- Code2Doc: Quality measurement and generation of C++ documentation

* Funded by DOE SciDAC RAPIDS2 (Platform Readiness) and LLNL's JACCEL project, in collaboration with 1) University of Minnesota, 2) Iowa State University, and 3) The University of North Carolina at Charlotte

Automating Legacy Code Modernization with Fortran2CPP

- Motivation
  - Fortran has powered scientific computing since the 1950s and remains widely used in HPC applications.
  - However, legacy Fortran code is hard to maintain, adapt, or integrate into modern systems.
  - Manual translation is slow; existing tools are brittle.
- Fortran2CPP: Translating Fortran to C++ more efficiently and reliably using LLMs



Multi-dialogue Dataset – Enabling Learning from Experiences

- Dialogue: A complete sequence of interactive communication between two or more agents, with a clear beginning and end, unified by a common context or purpose. It's composed of multiple dialogues/turns and represents the full scope of the interaction.
  - Turn: A single turn of exchange between agents, consisting of one utterance/message and its corresponding response. This forms the basic unit of interaction within a conversation.
- Why could multi-dialogue datasets be better (hypothesis) ?
  - Keep entire history of translation workflow
  - Embedding rich semantics from compilation and execution feedback
  - Both success and failed attempts are valuable data

Performance comparison of Fine-tuned models on HumanEval-Fortran2CPP

Evaluation Method	Before/After Fine-tuning	DeepSeek-Coder 6.7B	CodeLlama 13B	StarCoder 15.5B	GPT-4 Turbo
CodeBLEU Score	Original	0.072	0.090	0.067	0.203
	Fine-tuned with Code Pairs	0.098	0.101	0.089	
	Fine-tuned with Dialogue	0.225	**0.239**	0.225	
Compilation Check	Original	0.0 (0 / 126)	0.0 (0 / 126)	0.0 (0 / 126)	0.90 (113 / 126)
	Fine-tuned with Code Pairs	0.64 (81 / 126)	0.60 (75 / 126)	0.59 (74 / 126)	
	Fine-tuned with Dialogue	0.84 (106 / 126)	**0.92 (116 / 126)**	0.64 (81 / 126)	
Execution Test Evaluation	Original	0.0 (0 / 126)	0.0 (0 / 126)	0.0 (0 / 126)	**0.83 (104 / 126)**
	Fine-tuned with Code Pairs	0.61 (77 / 126)	0.46 (58 / 126)	0.43 (54 / 126)	
	Fine-tuned with Dialogue	0.71 (89 / 126)	0.74 (93 / 126)	0.51 (64 / 126)	
Manual Investigation	Original	0	3.75	0	**4.75**
	Fine-tuned with Code Pairs	3.75	3.2	3.4	
	Fine-tuned with Dialogue	4.7	**4.75**	4.7	

Conclusion

- A novel LLM-based approach featuring a dual-agent workflow for iterative dataset generation
  - The generated multi-turn dialogue dataset provides extensive knowledge from compilation and execution feedback
  - Experimental results demonstrated effectiveness across various evaluation metrics, outperforming traditional code-pair datasets
- This work makes a significant contribution to automating legacy code modernization and improving software portability in scientific computing.

Le Chen, Bin Lei, Dunzhi Zhou, Pei-Hung Lin, Chunhua Liao, Caiwen Ding, Ali Jannesari, Fortran2CPP: Automating Fortran-to-C++ Translation using LLMs via Multi-Turn Dialogue and Dual-Agent Integration, Jan. 2025
https://arxiv.org/pdf/2412.19770

CompilerGPT: Leveraging LLMs for Analyzing and Acting on Compiler Optimization Reports

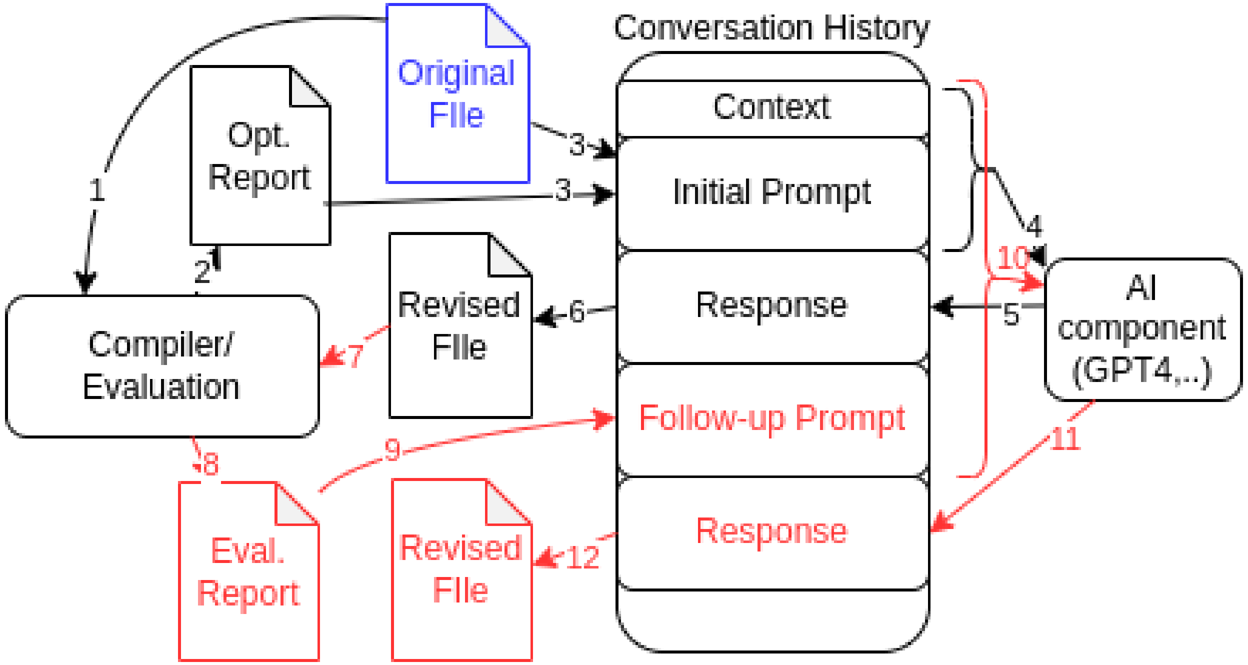
- Compiler reports (Clang, GCC) are cryptic, technical, and often overwhelming
- Developers struggle to interpret reports and act on them effectively
- Reports vary between compilers and are filled with low-level jargon (e.g., licm, regalloc)

An excerpt of a Clang/LLVM optimization report for simplematrix.cc, using clang version 18.1.8.

```
simplematrix.cc:19:18: remark: failed to move load with loop-invariant
address because the loop may invalidate its value [-Rpass-missed=licm]
  19 |        res(i,j) += lhs(i, k) * rhs(k, j);
     |                 ^
simplematrix.cc:19:18: remark: failed to hoist load with loop-invariant
address because load is conditionally executed [-Rpass-missed=licm]

simplematrix.cc:19:18: remark: failed to move load with loop-invariant
address because the loop may invalidate its value [-Rpass-missed=licm]

simplematrix.cc:18:7: remark: loop not vectorized [-Rpass-missed=loop-
vectorize]
  18 |      for (int k = 0; k < lhs.columns(); ++k)
     |      ^

simplematrix.cc:14:5: remark: 1 reloads 1.249999e+02 total reloads cost 4
folded reloads 8.124992e+02 total folded reloads cost 4 virtual registers
copies 5.312495e+02 total copies cost generated in loop [-Rpass-
missed=regalloc]
  14 |    for (int j = 0; j < res.columns(); ++j)
     |    ^
```

Solution: CompilerGPT

- A framework that integrates:
  - A compiler (Clang/GCC)
  - A large language model (GPT-4o or Claude Sonnet)
  - A user-defined evaluation harness
  - Iterative and automated analysis of optimization reports and code rewriting
- Use techniques:
  - Chain-of-thought
  - Testing
  - Negative prompting
  - Code snippet optimization



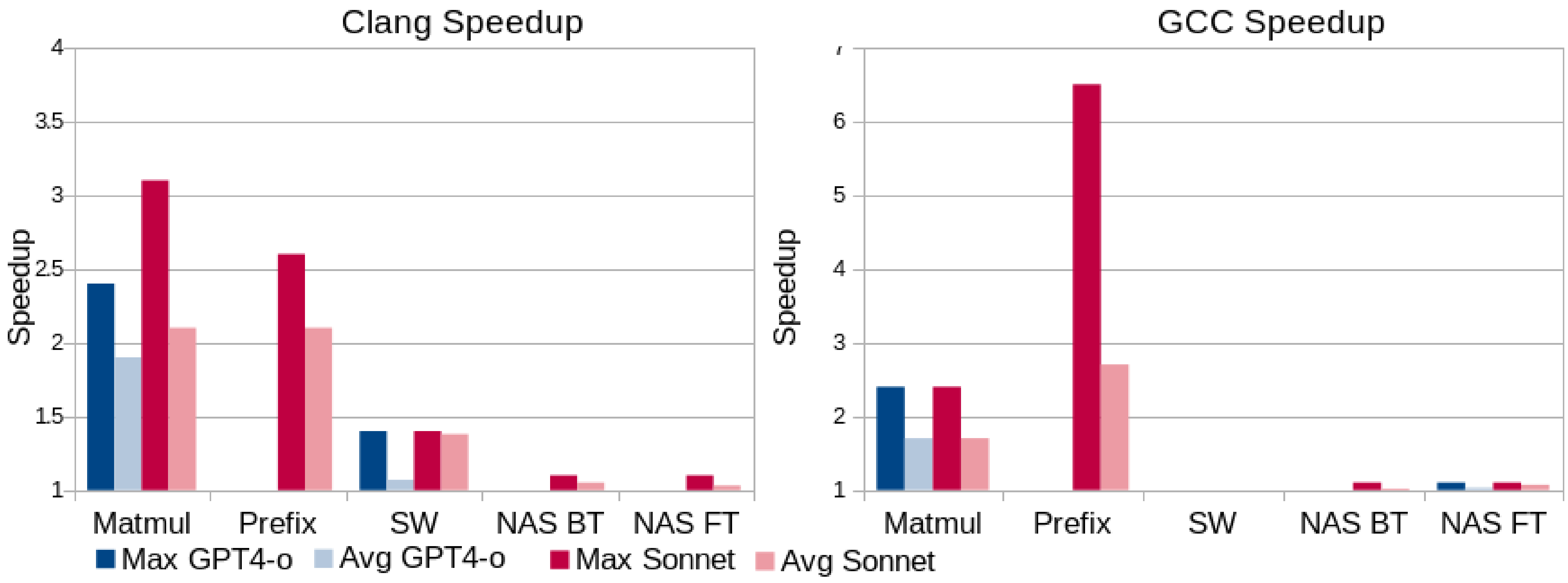
https://github.com/LLNL/CompilerGPT

Peter Pirkelbauer and Chunhua Liao, CompilerGPT: Leveraging Large Language Models for Analyzing and Acting on Compiler Optimization Reports, the Fifth Workshop on Compiler-assisted Correctness Checking and Performance Optimization for HPC, Held in Conjunction with ISC 2025, June 10-13, Hamburg, Germany

Prompt examples

ID	Prompt
Context	You are an expert in C++ compiler optimizations and code performance tuning for modern Intel x86.
First Prompt	You are provided with the following code snippet: {{code}}. The execution time for 10 runs of the code is {{scoreint}} milliseconds. The compiler, {{compilerfamily}}, has generated the following optimization report: {{report}}. Your goal is to focus on high-impact optimizations that significantly reduce execution time. Follow these tasks carefully: **Task 1**: Report Analysis - Analyze the optimization report and extract a prioritized list of the top 3 issues that are most likely to have a significant impact on performance. Focus on issues that are directly related to execution time bottlenecks or critical paths in the code. **Task 2**: Code Analysis - Based on the extracted prioritized list, select the single highest-impact issue. Identify the specific code segments that are directly related to this issue. Do not suggest changes to unrelated or low-impact parts of the code. **Task 3**: Code Improvement - Rewrite only the identified code segments from Task 2 to address the selected issue and enable better compiler optimizations. Ensure the rewritten code is functionally equivalent to the original code. Return the entire code in a single code block.
Prompt if success	The execution time for 10 runs of the latest code is {{scoreint}} milliseconds. *The full prompt continues like the first prompt and is omitted for brevity.*
Prompt on compiler Error	This version did not compile. Here are the error messages: {{report}}. Try again.
Prompt on failed tests	This version failed the regression tests. Here are the error messages: {{report}}. Try again.

Preliminary results



- Matmul: Sonnet optimizing Clang: 3.1x speedup, achieved through loop blocking, unrolling, and reducing register pressure.
- Prefix Scan: Sonnet optimizing GCC: 6.5x speedup, achieved by hoisting temporary vectors and switching to raw memory management.
- Smith-Waterman: Sonnet optimizing Clang: 1.4x speedup, achieved by privatizing local maximum variables and adding memory prefetch
- instructions.
- BT and FT: limited context window for larger kernels, variability in generated outputs across iterations, function call and OpenMP barrier overhead

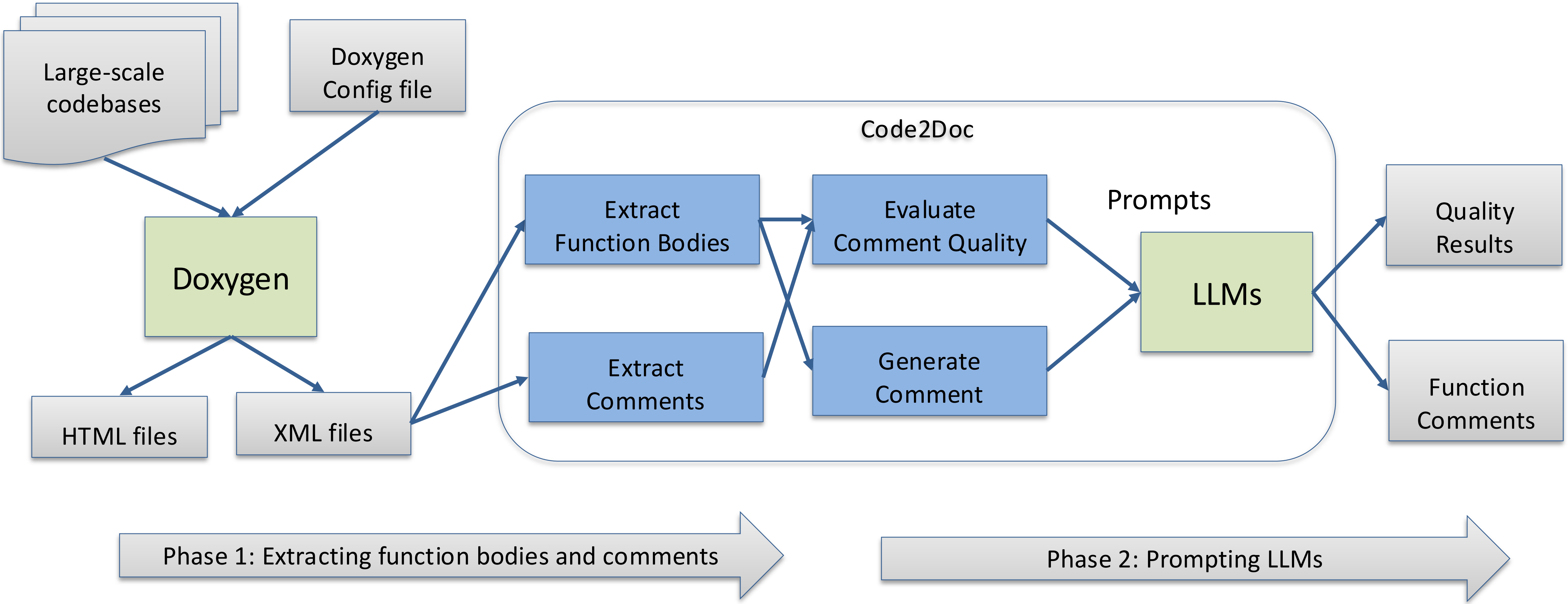
Summary

- LLMs can:
  - Summarize and prioritize compiler feedback
  - Rewrite code for better performance
- But they also:
  - Hallucinate
  - Need good tests
  - Struggle with large codebases
  - may not converge after multiple iterations
- Ongoing work:
  - Profile-guided region selection
  - Auto test generation
  - Learning from experiences: Fine-tuning and/or Retrieval Augmented Generation
  - More robust and consistent prompting

Code2Doc: Harnessing Generative AI for Software Documentation

- Motivation
  - Documentation forms a significant portion of modern codebases
  - Developers often avoid writing or maintaining documentation
  - As a result, many software projects suffer from incomplete or outdated documentation
- Generative AI models can produce convincing software source code given text prompts
  - is it possible to harness these models to invert the process to produce text descriptions given source code?
  - Can we also measure the quality of comments using LLMs?
  - What are the special considerations required when working with sensitive software source code?

Approach: a Pipeline using Doxygen and LLMs



Prompt 1: Measure the Quality of Comments

- **Objective**: Assess how well comments explain and enhance understanding of a C++ function.
- **Evaluation Criteria**: 8 aspects
  - **Functionality**: Clarity on what the function does.
  - **Input Parameters**: Explanation of input parameters.
  - **Side Effects**: Mention of side effects (e.g., global state changes).
  - **Return Value**: Description of the function's return value.
  - **Error Handling**: Explanation of error or exception handling.
  - **Consistency**: Alignment of comments with actual function behavior.
  - **Edge Cases**: Addressing potential edge cases.
  - **Sufficient Length**: Adequate detail and length of comments.
- **Scoring**:
  - Each criterion scored as 0 or 1: extract simplest answer from LLMs
  - Sum scores for a **Total Score:** simple addition for now, weighted sum is possible too.
- **Final Output**:
  - Provide a brief explanation for each score.
  - Suggest areas for improvement.
  - Present final score as: **Total Score: [X]** (replace [X] with numeric score).

Prompt 2: Generating Doxygen Comments for C++ Functions

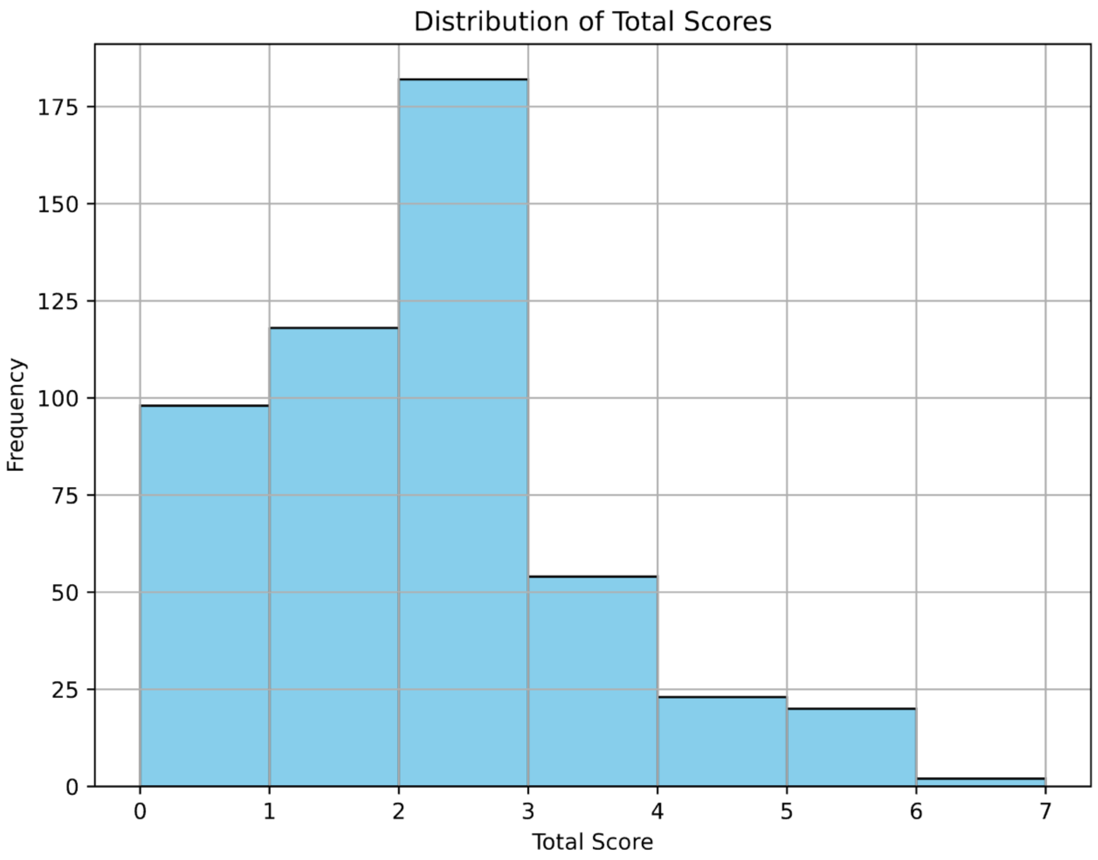
- **Task**: Create comprehensive Doxygen comments for a specified function.
- **Comment Sections to Include**:
  - **Purpose**: Describe the function's intent.
  - **Parameters**: Detail input parameters.
  - **Return Value**: Explain what the function returns.
  - **Warnings**: Note any important warnings.
  - **Preconditions**: State conditions that must be met before execution.
  - **Postconditions**: Describe conditions after execution.
  - **Side Effects**: Mention any side effects.
  - **Notes/Examples**: Provide additional relevant information or examples.
- **Requirements**:
  - Follow Doxygen formatting conventions.
  - Use plain text (no markdown syntax).
- **Output**: Provide the generated comments followed by the function signature.
  - Additionally call Doxygen to generate final html reference docs

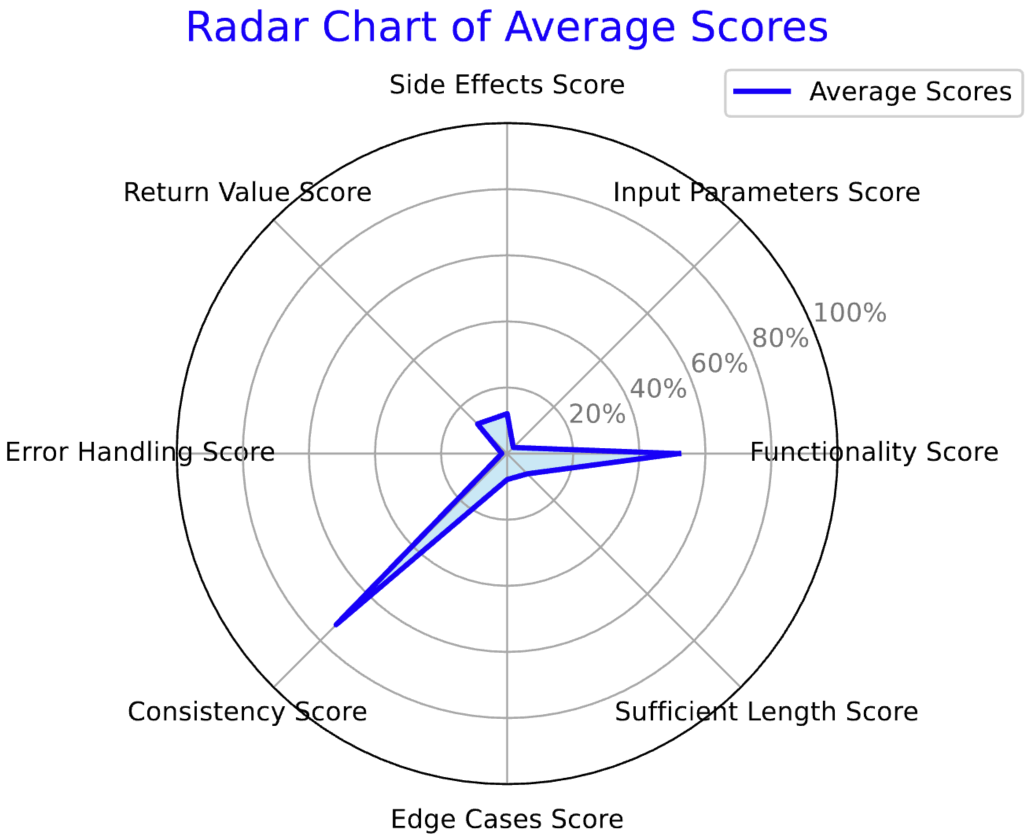
Experiment Settings

- Machine: Dell T2920 workstation
  - Intel(R) Xeon(R) Gold 6238 CPU @ 2.10GHz: Dual Processors
  - Main Mem.: 128 GB
  - NVIDIA RTX A6000: 48 GB
  - RHEL 8.10
- LLM: llama3.1 70B
  - 8-bit quantized version: 39 GB
  - Input context window size: 128k tokens
  - Served via Ollama: v 0.3.10
  - Temperature: 0.5
- Python 3.11 and extra packages
- ROSE v. 0.11.145.150
  - SageInterface namespace: important functions to process ROSE AST
  - 497 C++ functions

Package	Version	License	Purpose
Ollama	0.3.10	MIT	Serve Local LLMs
Llama	3.1	Llama 3.1	Local LLM
Matplotlib	3.9.3	PSF (BSD-compatible)	Visualization in python
Numpy	2.1.3	BSD 3-Clause	Operating on arrays in python
Openai	1.56.0	MIT	API interacting with LLMs
Pandas	2.2.3	BSD 3-Clause	Process tabular data
Reportlab	4.2.5	BSD for core	PDF report generation
ROSE	0.11.145.150	BSD 3-Cause	Example large scale C++ code base

Distribution of Total Scores of Comment Quality: ~500 SageInterface functions in the ROSE compiler (written in C++)





The quality ratings are **consistent with manual ratings in over 90% cases** for a sampled subset.

SageInterface

Main Page | Namespaces | Classes | Files | Examples | Search

- SageInterface
  - Namespaces
  - Classes
    - Class List
      - SageInterface
    - Class Index
    - Class Hierarchy
    - Class Members
  - Files
  - Examples

◆ annotateExpressionsWithUniqueNames()

void SageInterface::annotateExpressionsWithUniqueNames (SgProject * project)

Annotates expressions in an AST with unique names.

This function traverses the abstract syntax tree (AST) of a given SgProject and annotates each expression node with a unique name. The unique name is generated based on the expression's class name and other factors to ensure uniqueness within the project.

Parameters

project A pointer to an SgProject object representing the AST to be annotated.

Returns

None

Warning

This function modifies the input AST by adding new attributes to expression nodes. It does not perform any checks for existing annotations or handle potential naming conflicts.

Precondition

The input project must be a valid, non-null pointer to an SgProject object. The project's AST should be properly constructed and contain at least one expression node.

Postcondition

After executing this function, each expression node in the input project's AST will have a new attribute named "UniqueNameAttribute" containing a unique name generated based on the expression's class name.

Note

This function uses a recursive traversal approach to visit all nodes in the input AST. It assumes that the input project's AST is well-formed and does not contain any cycles or invalid node references.

Quality of Generated Documents

ID	Purpose	Parameters	Return	Warning	Pre cond.	Post cond.	Example	Notes
0	1	1	1	1	1	1	0	1
55	1	1	1	1	1	1	0	1
110	1	1	1	0	1	1	0	1
165	1	1	1	0	1	1	1	1
220	0	0	1	0	1	1	1	1

- For simplicity, only give score of 0 or 1: 1 means getting it right: Overall **32/40 correct (80%)**
  - ID 0: example does not store the returned value to a variable
  - ID 55: example uses the namespace with a dot expression
  - ID 110:
    - warning, read into printf() warning message within #if 0 … #endif
    - Two examples, one does not use return value, only one is enclosed within \code … \endcode
  - ID 165: warning, it says undefined behavior for Null input, in reality it will trigger assertion failure.
  - ID 220:
    - used a wrong term: initial name vs. initialized name in both purpose and parameters comments
    - warning, it says undefined behavior for Null input, in reality it will trigger assertion failure.

Summary

- Using locally deployable LLMs for comments generation is promising (as a baseline):
  - Automate both comment rating and doc generation at scale: ~15 seconds response time per prompt
  - Grading comment quality: over 90% correct for a sampled subset, of ROSE's SageInterface functions
  - Generating comments: around 80% correct for a sampled subset
  - Open-weight LLMs are still behind commercial models: partially due to quantization

- Ongoing work to improve results and address hallucinations and randomness
  - Input
    - Provide complete input (return types, function bodies, callees via call graphs)
    - Remove distractions: stale comments, #if 0 .. #endif dead code blocks
    - Sanity check: ask LLMs to check input and refuse the request when needed
  - Output:
    - Iterative self-critique or third-party critique, until results are satisfactory
    - Generate unit tests based on docs, then test the original functions.
    - Cross-check with static analysis
    - Human-in-the loop: add labels or ratings as feedback to fine tune the models
  - Models: try better open-weight models
    - without compression, using larger GPU machines if possible

A few other AI Tools (demos)

- DeepWiki
- VS Code + Cline + LLMs
  - Help reproduce a paper
  - Help proposal writing
- alphaXiv (if time permits)

Security Moment

Using AI models safely

Be mindful of what you're sharing

- Who has access to this data?
- Is this data leaving LLNL or U.S. soil?
- What could *they* do with all LLNL queries?

In general

- Would you feel comfortable searching for your prompt in Google knowing everyone can see it?

****Publicly releasable information only****

Ethical Considerations

AI is a powerful tool that must be balanced with ethical considerations, such as:

- Data privacy
- Copyright issues
- Safety
- Fairness
- Misinformation/misinterpretations

If you have questions or are seeking guidance…

- Reference LLNL AI Policy (CSP Policy 2800) created with help from the AI Community of Practice

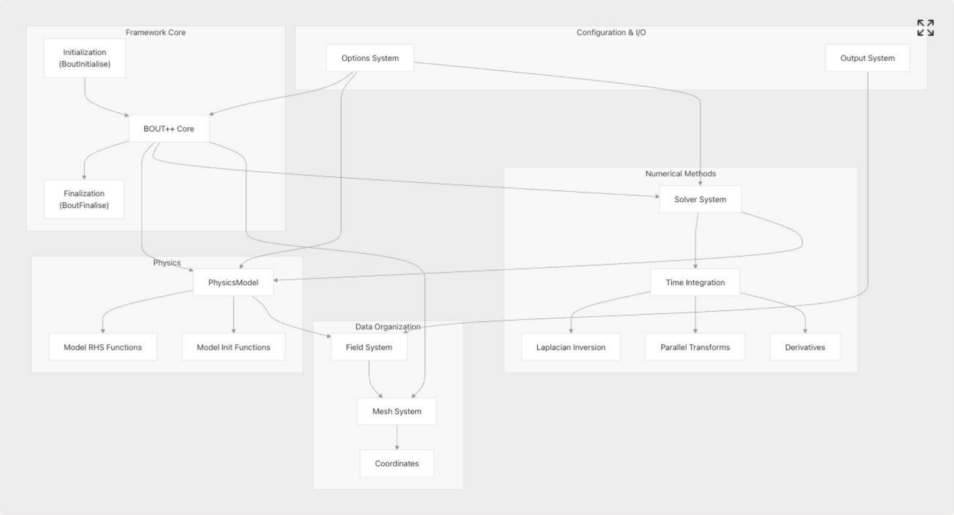
DeepWiki boutproject/BOUT-dev

powere

Last indexed: 28 April 2025 (c18ef3)

Framework Architecture

BOUT++ is structured around several core components that work together to handle different aspects of plasma simulations. The framework adopts an object-oriented approach, with a clean separation between physical models, numerical methods, and I/O facilities.



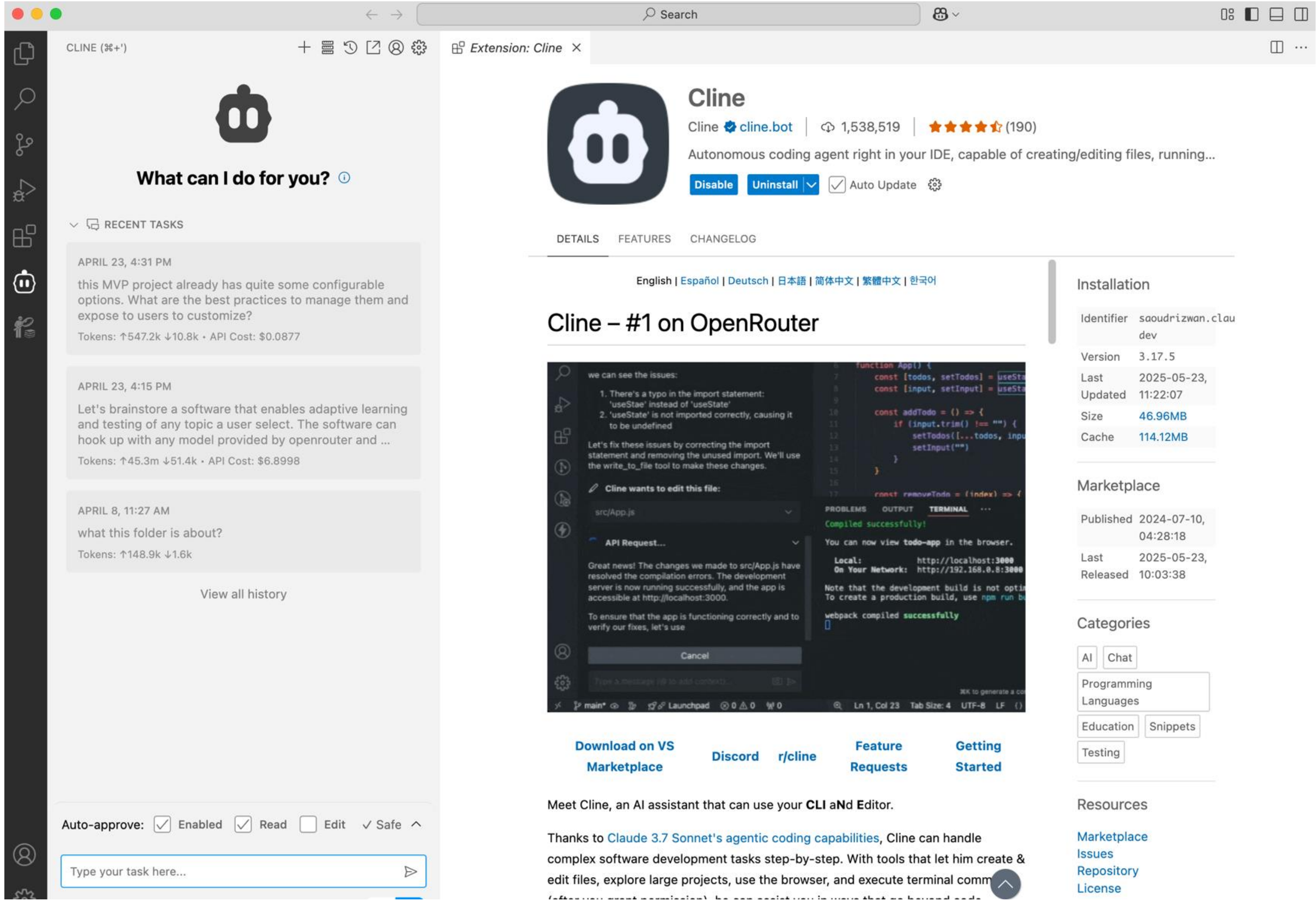
Sources: `src/bout++.cxx` 130–214 `include/bout/physicsmodel.hxx` 35–123 `include/bout/solver.hxx` 75–206 `CMakeLists.txt` 85–190

Ask Devin about boutproject/BOUT-dev

Deep Research

https://deepwiki.com/boutproject/BOUT-dev

A “Production” IDE: MS VS Code + Cline + LLM APIs



Demo: Coding using Cline

- Goal: replicate software infrastructure from Google Co-scientist paper
  - Start from scratch: from MVP to more feature-rich versions
  - Demo of one existing system coded using Cline

- Workflow
  - Configuring things: API keys, models
  - Tell Cline what you want
  - Watch and confirm Cline's actions
  - Checking results
  - Fix any errors
  - Repeat!

Reproduced functionalities so far

Welcome to the AI Co-Scientist System

Set your research goal and run cycles to generate hypotheses.

Research Goal:
make human live to 1000 years

Advanced Settings

LLM Model: openai/o3

Number of Hypotheses per Cycle: 3

Generation Temperature (Creativity): 0.7

Reflection Temperature (Analysis): 0.5

Elo K-Factor (Ranking Sensitivity): 32

Top K for Evolution: 2

Submit Research Goal | Run Next Cycle

Instructions: Enter a research goal and click "Submit Research Goal" to start a new process. Click "Run Next Cycle" to perform another iteration on the current set of hypotheses.

Submit a research goal to begin.

Results

Errors

Step: ranking2

Hypotheses:

- **Hypothesis 3: Enhanced Mitochondrial Turnover and Function through Targeted Gene Therapy of Key Mitochondrial Biogenesis Factors** (ID: G1874, Elo: 1269.30)

  Rationale: Mitochondrial dysfunction is a hallmark of aging. We hypothesize that enhancing mitochondrial turnover (mitophagy) and function through targeted gene therapy can improve cellular energy production and reduce oxidative stress, ultimately extending lifespan. This would involve using viral vectors to deliver genes encoding key transcription factors involved in mitochondrial biogenesis (e.g., PGC-1α) and mitophagy (e.g., PINK1, Parkin) specifically to tissues with high metabolic demand (e.g., brain, muscle, heart). By boosting mitochondrial health, we aim to improve cellular resilience and delay age-related decline.

  Novelty: MEDIUM

  Feasibility: MEDIUM

- **Hypothesis 4: Development of a Circulatory System Modulation System for Perpetual Youth** (ID: G5807, Elo: 1241.13)

  Rationale: The circulatory system delivers nutrients and removes waste products, critical functions that decline with age. We hypothesize that a system for perpetually modulating the circulatory system – including blood composition, vessel elasticity, and blood flow – can significantly extend lifespan. This could involve periodic blood transfusions with optimized blood products (e.g., enriched with specific growth factors or lacking pro-inflammatory cytokines), coupled with interventions to maintain vascular health such as gene therapy targeting endothelial cell function or intermittent pneumatic compression to improve circulation. The goal is to maintain a 'youthful' circulatory environment, promoting tissue regeneration and delaying age-related diseases.

  Novelty: MEDIUM

  Feasibility: MEDIUM

- **Combined: Hypothesis 1: Periodic Cellular Senescence Reversal via Targeted Delivery of Senolytics and Senomorphics & Hypothesis 4: Development of a Circulatory System Modulation System for Perpetual Youth** (ID: E2284, Elo: 1225.69)
  Parents: G7304, G5807

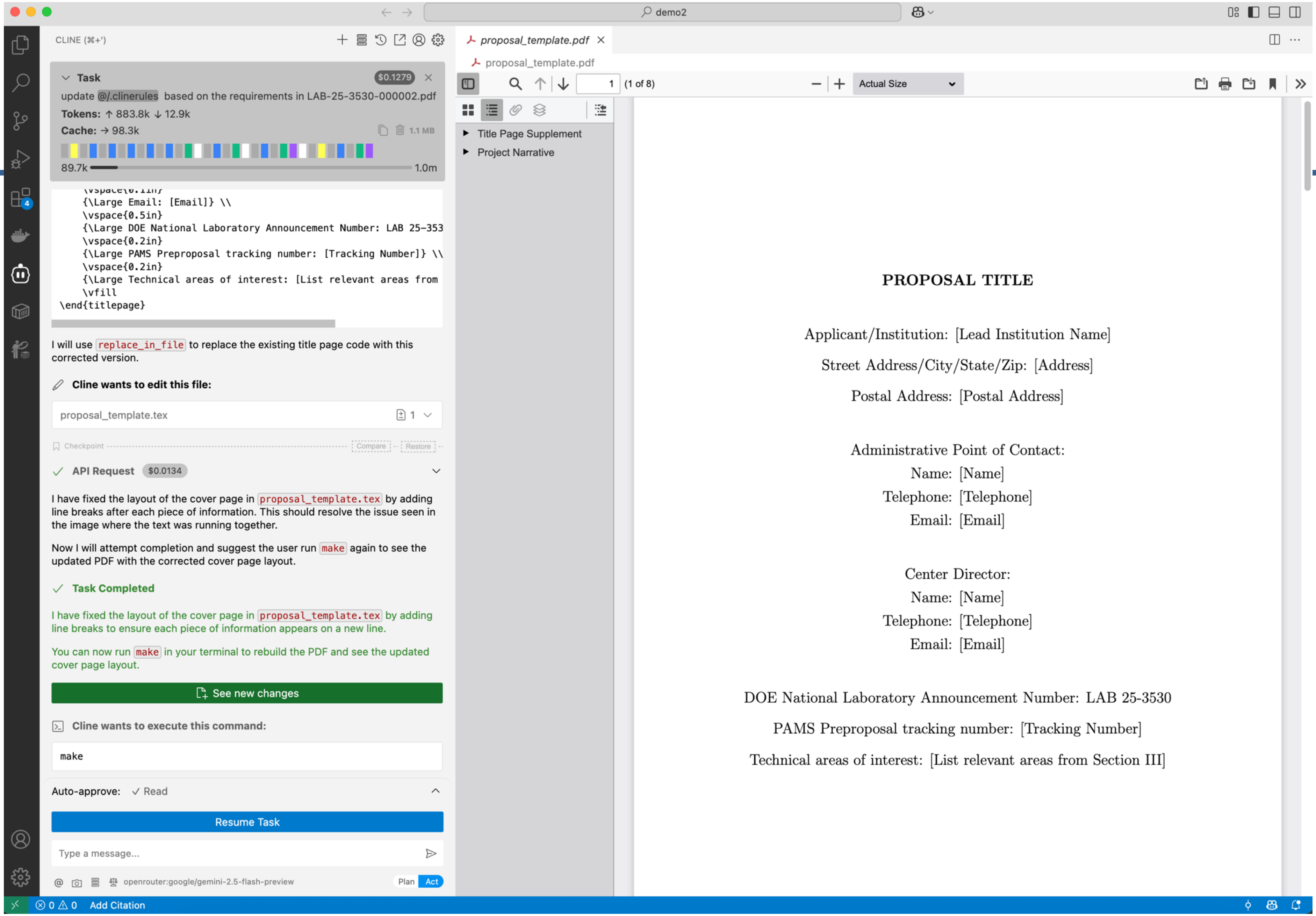
  Combination of: 1. Rationale: Cellular senescence accumulation is a major driver of aging. We hypothesize that periodically reversing senescence through targeted delivery of senolytic drugs (killing senescent cells) and senomorphic drugs (modifying senescent cell function) can significantly extend lifespan. Specifically, we propose using nanoparticle-based delivery systems that are triggered by specific biomarkers associated with aging (e.g., increased levels of certain inflammatory cytokines or altered extracellular matrix composition) to release senolytics and senomorphics directly into senescent tissues. This minimizes systemic toxicity and maximizes efficacy, potentially resetting cellular aging processes and extending lifespan. 2. Rationale: The circulatory system delivers nutrients and removes waste products, critical functions that decline with age. We hypothesize that a system for perpetually modulating the circulatory system – including blood composition, vessel elasticity, and blood flow – can significantly extend lifespan. This could involve periodic blood transfusions with optimized blood products (e.g., enriched with specific growth factors or lacking pro-inflammatory cytokines), coupled with interventions to maintain vascular health such as gene therapy targeting endothelial cell function or intermittent pneumatic compression to improve circulation. The goal is to maintain a 'youthful' circulatory environment, promoting tissue regeneration and delaying age-related diseases.

  Novelty: MEDIUM

  Feasibility: MEDIUM

Demo: Help Proposal Writing

- Goal: create a complete skeleton latex template complying with FOA
- Workflow
  - Download a FOA pdf
  - Draft a default proposal .clinerules
  - Update .clinerules based on the FOA pdf
  - Generate latex template that can build a proposal
  - Generate makefile
  - Fix build errors
  - Fix any pdf rendering issues


demo2
CLINE (⌘+')
Task
$0.1279
update @/.clinerules based on the requirements in LAB-25-3530-000002.pdf
Tokens: ↑ 883.8k ↓ 12.9k
Cache: → 98.3k
1.1 MB
89.7k
1.0m
{\Large Email: [Email]} \\
\vspace{0.5in}
{\Large DOE National Laboratory Announcement Number: LAB 25-353
\vspace{0.2in}
{\Large PAMS Preproposal tracking number: [Tracking Number]} \\
\vspace{0.2in}
{\Large Technical areas of interest: [List relevant areas from
\vfill
\end{titlepage}
I will use replace_in_file to replace the existing title page code with this corrected version.
Cline wants to edit this file:
proposal_template.tex
Checkpoint
Compare
Restore
API Request
$0.0134
I have fixed the layout of the cover page in proposal_template.tex by adding line breaks after each piece of information. This should resolve the issue seen in the image where the text was running together.
Now I will attempt completion and suggest the user run make again to see the updated PDF with the corrected cover page layout.
Task Completed
I have fixed the layout of the cover page in proposal_template.tex by adding line breaks to ensure each piece of information appears on a new line.
You can now run make in your terminal to rebuild the PDF and see the updated cover page layout.
See new changes
Cline wants to execute this command:
make
Auto-approve: ✓ Read
Resume Task
Type a message...
openrouter:google/gemini-2.5-flash-preview
Plan
Act
Add Citation
proposal_template.pdf
proposal_template.pdf
(1 of 8)
Actual Size
Title Page Supplement
Project Narrative
PROPOSAL TITLE
Applicant/Institution: [Lead Institution Name]
Street Address/City/State/Zip: [Address]
Postal Address: [Postal Address]
Administrative Point of Contact:
Name: [Name]
Telephone: [Telephone]
Email: [Email]
Center Director:
Name: [Name]
Telephone: [Telephone]
Email: [Email]
DOE National Laboratory Announcement Number: LAB 25-3530
PAMS Preproposal tracking number: [Tracking Number]
Technical areas of interest: [List relevant areas from Section III]

Title Page Supplement

Table 1: Center Members

Last Name	First Name	Title	Institution Name
[Last Name]	[First Name]	[Title]	[Institution Name]
[Last Name]	[First Name]	[Title]	[Institution Name]

Table 2: Summary Budget Information by Institution ($ in thousands)

Institution Name	Lead PI Last Name	Lead PI First Name	Year1 Budget ($K)	Year2 Budget ($K)	Year3 Budget ($K)	Year4 Budget ($K)	Year5 Budget ($K)
[Institution Name]	[Last Name]	[First Name]	[First Name]	[Budget]	[Budget]	[Budget]	[Budget]
[Institution Name]	[Last Name]	[First Name]	[Budget]	[Budget]	[Budget]	[Budget]	[Budget]
Total Budget			[Total]	[Total]	[Total]	[Total]	[Total]

Table 3: Summary Budget Information by Major Thrust Area ($ in thousands)

Major Thrust Area	Lead for Major Thrust Area	Year1 Budget	Year2 Budget	Year3 Budget	Year4 Budget	Year5 Budget
[Major Thrust Area]	[Lead Name]	[Budget]	[Budget]	[Budget]	[Budget]	[Budget]
[Major Thrust Area]	[Lead Name]	[Budget]	[Budget]	[Budget]	[Budget]	[Budget]
Total		[Total]	[Total]	[Total]	[Total]	[Total]

Contents

BOUT++: a framework for parallel plasma fluid simulations

B.D.Dudson *, H.R.Wilson
Department of Physics, University of York, Heslington, York YO10 5DD, UK

M.V.Umansky, X.Q.Xu
Lawrence Livermore National Laboratory, Livermore, CA 94551, USA

P.B.Snyder
General Atomics, P.O. Box 85608, San Diego, CA 92186-5608, USA

Abstract

A new modular code called BOUT++ is presented, which simulates 3D fluid equations in curvilinear coordinates. Although aimed at simulating Edge Localised Modes (ELMs) in tokamak x-point geometry, the code is able to simulate a wide range of fluid models (magnetised and unmagnetised) involving an arbitrary number of scalar and vector fields, in a wide range of geometries. Time evolution is fully implicit, and 3^{rd}-order WENO schemes are implemented. Benchmarks are presented for linear and non-linear problems (the Orszag-Tang vortex) showing good agreement. Performance of the code is tested by scaling with problem size and processor number, showing efficient scaling to thousands of processors.

Linear initial-value simulations of ELMs using reduced ideal MHD are presented, and the results compared to the ELITE linear MHD eigenvalue code. The resulting mode-structures and growth-rate are found to be in good agreement ($\gamma_{BOUT++} = 0.245\omega_A$, $\gamma_{ELITE} = 0.239\omega_A$, with Alfvénic timescale $1/\omega_A = R/V_A$). To our knowledge, this is the first time dissipationless, initial-value simulations of ELMs have been successfully demonstrated.

Key words: Plasma simulation, curvilinear coordinates, tokamak, ELM
PACS: 52.25.Xz, 52.65.Kj, 52.55.Fa

1. Introduction

BOUT++ is a new highly adaptable, object-oriented C++ code for performing parallel plasma fluid simulations with an arbitrary number of equations in 3D curvilinear coordinates using finite-difference methods. It has been developed from the original **BOU**ndary **T**urbulence 3D 2-fluid tokamak edge simulation code BOUT [1,2,3,4,5,6], borrowing ideas and algorithms, but has been substantially altered and extended. Though designed to simulate tokamak edge plasmas efficiently, the methods used are very general and can be adapted to many other situations: any coordinate system metric tensor $g^{ij} = g^{ij}(x, y)$ (i.e. constant in one dimension) can be specified, which restricts the coordinate system to those with axi- or translationally symmetric geometries. Even 2D metric tensors encompass a wide range of situations, allowing the code to be used to simulate plasmas in slab, sheared slab, cylindrical and non-orthogonal coordinate systems such as flux coordinates for tokamak simulations. Extension of the code to allow 3D metric tensors would be relatively straightforward, but is not currently necessary for the problems considered here.

* Corresponding author.
Email address: bd512@york.ac.uk (B.D.Dudson).

Preprint submitted to Computer Physics Communications *30 October 2018*

https://www.alphaxiv.org/abs/0810.5757

Conclusion

- Many AI tools exist and under development
  - https://lmarena.ai/
  - https://livebench.ai/#/
  - https://openrouter.ai/
  - https://www.swebench.com/
- Limitations
  - Hallucinations: e.g. non-existing references
  - Context window limit
  - Costs: API can be expensive
  - Suboptimal decisions, using old version APIs
  - Chasing its own tail: does not converge
- Solutions
  - Explore novel datasets, training methods, and workflows (e.g. learning from experiences)
  - Test-driven and be specific
  - Experiment with different AI models
  - Ask and Learn, then tailor decisions

Coding	Research and Writing
• Github Copilot, OpenAI Codex, Claude Code, Google Jules • VS Code + Cline extension • Cursor, Windsurf • Devin, OpenHands • Algorithm discovery and optimization: AlphaEvolve • Documentation generation: DeepWiki, • Research Prototypes: Fortran2Cpp, CompilerGPT, Code2Doc	• Reading papers: alphaXiv • Literature Review: OpenAI Deep Research, similar feature from Grok, Gemini, and Perplexity • Hypothesis Generation: Google AI Co-Scientist • Error checking: YesNoError • Fully-Automated: AI Scientist, AI Researcher, Agent Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC