Title: NSAM4SCI: Neuro-Symbolic Autoresearch Agents for Scientific Code Development

Abstract

Scientific computing relies on large, complex codebases where correctness is paramount but development is slow, verification is manual, and surrogate models sacrifice interpretability for speed. We propose NSAM4SCI, a framework that combines LLM-based agentic workflows with Neuro-Symbolic Abstract Machines (NSAMs) to enable verified scientific code generation, optimization, and surrogate construction.

NSAMs are neural networks structurally equivalent to programming language interpreters, enabling principled compilation of symbolic programs into neural architectures and decompilation back to interpretable code. LLM agents bridge the gap between real-world scientific codebases and the declarative representations NSAMs require, providing code comprehension, translation, and orchestration capabilities.

We will demonstrate this unified framework on three classes of public DOE-relevant artifacts: (1) scientific kernels, where known structure is preserved and uncertain components are learned as constrained surrogates; (2) scientific workflows, where extracted structure supports analysis, repair, and policy learning; and (3) program-logic tasks such as verified optimization or synthesis, where symbolic structure enables checkable transformations. Our approach treats code as data under a common representation that supports multiple forms of scientific software reasoning (code understanding, constrained learning, and trustworthy transformation) within one neuro-symbolic agent framework. Phase II will scale to full DOE simulation codes and integrate with the American Science Cloud.

Background:
DOE's large scientific codes are critical research infrastructure, but they remain difficult to translate, verify, port, optimize, and modernize with current AI tools. LLM agents can assist with syntax and local edits, but they do not reliably preserve scientific semantics, modular structure, or numerical constraints required for trustworthy scientific software[1][2]. **We address this gap by converting scientifically meaningful code fragments into typed intermediate representations that support verification, interpretable transformation, and selective learned modernization of legacy scientific kernels.** An example value proposition is demonstrated in **Figure 1**. The example chosen is subsurface system transport modeling[3] but the principle is generalizable. Traditional attempts to 'learn' these codes involve training surrogate models on outputs, but this discards rich symbolic representation in the manually crafted codes that encapsulates invaluable deep scientific expertise and understanding.

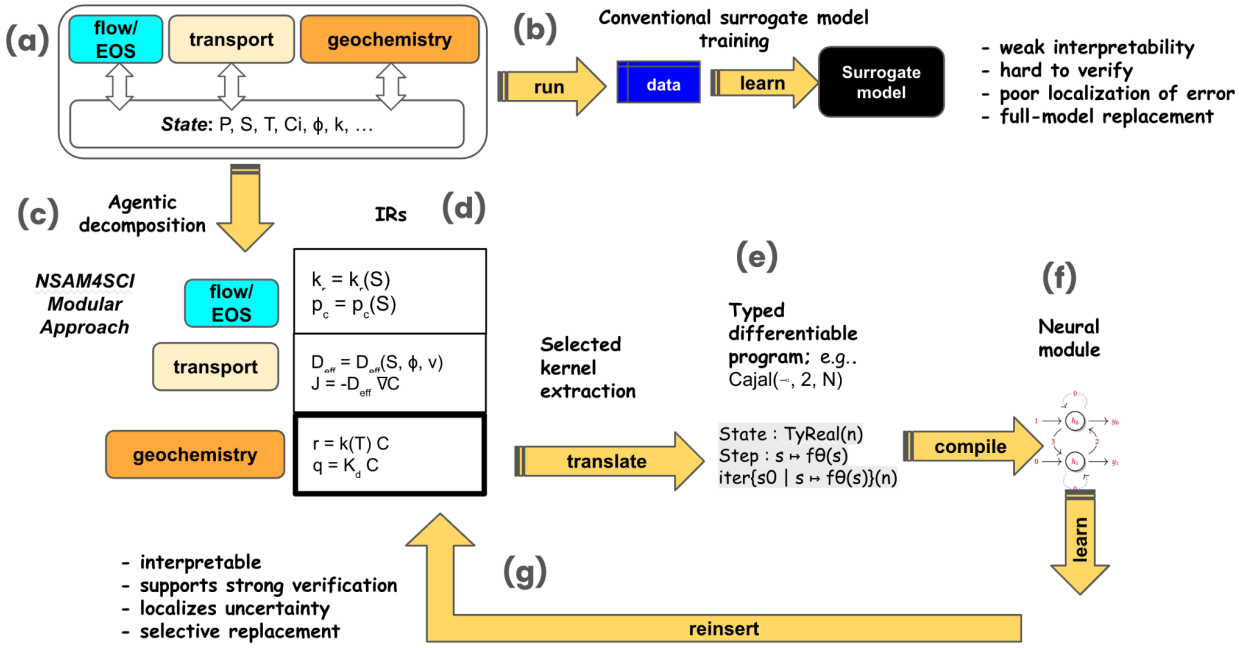


Figure 1. How NSAM4SCI can be applied to complex simulation codes, contrasting with traditional approaches **(a)** Example codes (here, for coupled reactive transport, but other complex codes can substitute) **(b)** Conventional surrogate model training: code is run to generate data for training a black-box surrogate; predictions lack explainability. **(c, d)** agentic extraction from codes to intermediate representations; here, agent identifies selected constitutive or reaction kernels inside the larger simulator **(e)** translation into a typed differentiable program (here, only the geochemistry kernel is shown), and **(f)** compiles them into a recurrent neural module. This enables selective learning or verification of scientifically meaningful components while leaving the broader solver scaffold intact, followed by reintegration into the larger solver scaffold **(g).**

We will develop **Neuro-Symbolic Abstract Machines for Scientific Codes Integration** (NSAM4SCI). This architecture will integrate with the Genesis platform by treating extracted code fragments and workflow artifacts as plug-and-play structured assets that can be indexed, reused, and evaluated alongside other mission capabilities. NSAM4SCI will enable the modular symbolic structures woven through scientific codes to be extracted and encoded as first-class

objects for AI systems, and allow the structure and dependencies of complex workflows to be encoded and learned (**Fig 1c-g)**. We will first use agentic methods to map subsets of codes to high-level declarative intermediate representations (IRs). These IRs are amenable to classic logical reasoning and proof checking, which carries many benefits in its own right: IRs can be formally verified, which is particularly important for automated program translation. We will go further than this: fragments of these IRs will then be translated to *differentiable programs*, which can be compiled directly into neural architectures, using methods our team has created[4]. Additionally, these IRs can also be used for deriving explanations from black-box surrogate models, with applications across the DOE science area, in any domain where predictions are amenable to symbolic mechanistic explanation. Unlike full-model emulation, NSAM4SCI focuses on selective learning of uncertain kernels, closures, and interaction rules inside an explicitly structured scientific program or workflow. The goal is not to replace an entire multiphysics code with a black-box surrogate, but to preserve the trusted scientific mechanistic scaffold while learning only the parts that are uncertain, expensive, scale-dependent, or poorly specified.

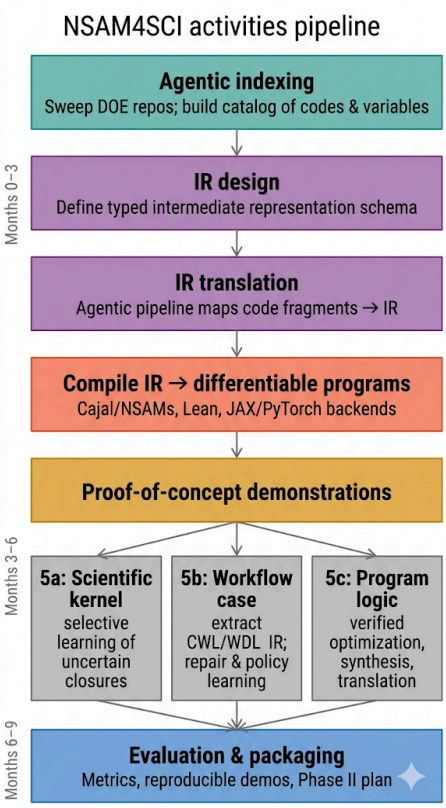
We will build on Cajal[4], a typed functional language developed by co-I Amin and collaborators whose programs compile exactly to recurrent neural networks. We will also draw on our work on agent-based scientific program synthesis, and on the use of logic programs as substrates for programmatic IRs[5]. We recently developed a method for distilling deep learning models into interpretable programs, and applied this to the task of generating chemical classification programs from deep learning chemical classifiers [6].

Project Objectives: NSAM4SCI addresses the Genesis focus area on trustworthy, domain-relevant AI capabilities for science and engineering by enabling AI systems to work with scientific codes and workflows as structured artifacts rather than only as text or black-box executables. The Phase I objectives are: (1) to develop a core intermediate representation (IR) for selected fragments of scientific codes, workflows, and program-logic tasks; (2) to build an agentic extraction pipeline that identifies and translates scientifically meaningful fragments into this IR; (3) to map selected IR fragments either to differentiable neuro-symbolic backends or to verification-oriented backends, depending on artifact structure; and (4) to demonstrate, on public DOE-relevant artifacts, that this approach provides measurable advantages in interpretability, constraint preservation, and trustworthy code transformation relative to less structured baselines. By the end of Phase I, we aim to deliver a working prototype stack for extraction, IR construction, backend routing, and evaluation, together with a clear path to Phase II deployment on larger DOE software systems.
Data Sources and AI Models. Phase I will use public DOE-relevant scientific software artifacts from DOECODE (https://www.osti.gov/doecode/) rather than private or proprietary data. On the AI side, the project will use two complementary model families: agentic large language model systems for extraction and translation of code fragments into the restricted IR, and neuro-symbolic differentiable-program frameworks based on Cajal/NSAM methods for structured learning over selected extracted kernels. Initial differentiable backends will use PyTorch and JAX, with Cajal as the starting compiler substrate. **AI Advantage**: The AI advantage lays in the ability to extract semantically meaningful components from existing codes from a typed IR, and use that representation to verify, translate, optimize, and selectively

modernize code fragments while preserving the larger scientific scaffold, rather than in constructing larger black-box surrogates from them. This directly addresses Genesis Topic 18-C and 18-E by advancing neuro-symbolic agents for code development and trustworthy AI for scientific software, with a Phase I focus on validated transformation of scientific kernels and portability to accelerator-oriented backends.

Proposed Research and Methods

Workflow: In Phase I, the target workflow is: identify a scientifically meaningful kernel inside a legacy code, extract it into a typed IR, verify preserved constraints, map it to a structured backend, and compare the result against fixed-mechanistic and less-structured baselines for fidelity, interpretability, and constraint preservation.



Activity 1: Agentic indexing of DOE scientific codes: Build a semantically initial indexed corpus of relevant software repositories and workflows from DOECODE and other relevant public sources, automatically extracting high-level structures, code relationships, and candidate fragments for downstream IR translation and search[7]. We will leverage our existing tools for agentic extraction[8], and with extracting latent ontologies from complex simulations such as EcoSIM[9] as a part of an LDRD[10].

Activity 2: IR design: We will define restricted IRs capable of representing the semantic core of three artifact classes: scientific kernels, scientific workflows, and program-logic tasks such as optimization or translation. The IR will capture typed interfaces, dependencies, guarded control flow, selected invariants, and annotations suitable for downstream learning or verification. Rather, it is a common schema for extracted fragments that are rich enough to support backend routing, constraint checking, and cataloging across multiple artifact classes. The Amin group has extensive experience in formal language design.

Activity 3: Automated IR Translation We will develop an agentic pipeline for automatic translation of fragments of indexed codes (activity 1) into the IR. The Mungall group has extensive experience with harness engineering for AI agents in extraction and translation tasks, and will lead the agentic development, with the Amin group leading formal validation.

Activity 4: Compilation of IR to differentiable programs We will use the existing Cajal NSAM system as the base system for differentiable programs. We will extend the expressivity of the Cajal system, including adding reals as first-class types so that scientific state vectors and update rules can be represented directly rather than through ad hoc encodings, and implement GPU-based optimizations together with compilation to JAX as well as the existing PyTorch backend for improved accelerator portability.

Activity 5: Three proof-of-concept demonstrations We will center Phase I evaluation on one scientific-kernel case directly aligned with DOE scientific software modernization, with smaller

workflow and program-logic demonstrations included to test generality of the IR and backend-routing approach.

5.1. Scientific-kernel case. Selected public scientific kernels, likely Fortran- or C-based, will be extracted into the IR and mapped to a constrained learning backend. Candidate kernels will be chosen where the surrounding scientific scaffold is trusted, the uncertain component is localized, relevant traces or benchmark data are available, and the resulting learned component can be reinserted and evaluated against mechanistic and surrogate baselines. Likely examples include constitutive or coupling laws in reactive transport, such as relative-permeability, sorption, reaction-rate, or porosity-permeability update terms. The goal is to preserve known structure while learning only selected uncertain components and to recover interpretable symbolic forms where possible.

5.2. Workflow case. A workflow artifact, such as a CWL, WDL, Nextflow, Parsl, or related pipeline, will be extracted into the IR and used for analysis, repair, or augmentation with a learned policy such as resource prediction, retry prediction, or execution-policy ranking.

5.3. Program-logic case. A code-logic task such as verified optimization, specification-driven synthesis, or translation will be represented in the IR and passed through a checkable transformation pipeline, producing either an interpretable optimization rule, a verified rewrite, or a constrained translation into a target language.

Activity 6: Evaluation, packaging, and deployment: Evaluation will combine shared and task-specific metrics. Shared metrics will include extraction fidelity, preservation of declared constraints, degree of interpretability, portability across artifact types, and the extent to which a common IR supports multiple backend realizations. Task-specific metrics will include surrogate fidelity for the kernel case, successful linting or repair outcomes for the workflow case, and correctness or equivalence measures for optimization, synthesis, or translation. For the kernel case, evaluation will explicitly compare IR-guided selective learning against two natural baselines: a fixed mechanistic model with no learned component, and a less structured surrogate that replaces a larger subsystem or component. This comparison will clarify when kernel-level learning actually provides an advantage in fidelity, interpretability, portability, or constraint preservation. All demonstrations will be built on public artifacts with reproducible evaluation pathways. Even at Phase I scale, this will establish whether scientific-code fragments can be extracted, represented, learned over, and verified in a way that is portable across artifact classes and realistic enough to justify transition to larger DOE software systems in Phase II.

The central technical question is whether one IR-first architecture can support multiple forms of scientific software reasoning without collapsing them into a single representation or backend. Phase I will therefore evaluate not only whether each demonstration works in isolation, but whether extraction, constraint preservation, backend selection, and verification can be shared across kernels, workflows, and program-logic tasks. This activity is a shared task across the project team, covering evaluation, packaging, dissemination, and Phase II planning.

Milestones and Decision Gate Metrics. The Phase I deliverable is a reusable prototype stack for extraction, restricted IR construction, backend routing, and evaluation on three public DOE-relevant artifact classes.

Months 0-3: Representation and ingestion

- *Define the Phase I restricted IR*, including typed interfaces, dependencies, selected invariants, and mappings to at least one learning backend and one verification backend.
- *Implement at least two extraction front ends* for public DOE-relevant artifacts, for example one code-oriented front end and one workflow-oriented front end.
- *Populate an initial catalog of extracted artifacts and metadata* suitable for AmSC-style indexing, demonstrating the “code as data” concept on a small but real corpus.
- *Identify at least two backend families for Phase I evaluation*, including one NSAM-style recurrent backend and one alternative structured backend suited to non-kernel artifacts.

Months 3-6: Proof-of-concept demonstrations and decision gate (go/no-go)

- *Demonstrate one scientific-kernel case* in which extracted symbolic structure is preserved and only selected uncertain components are learned, and compare this selective-learning approach against fixed mechanistic and broader surrogate baselines.
- *Demonstrate one workflow case in which a workflow is extracted into the IR* and then analyzed, repaired, or augmented with a learned policy such as resource or retry prediction.
- *Demonstrate one code-logic case*, such as verified optimization, synthesis, or translation, in which extracted symbolic structure supports interpretable and checkable transformations.
- *Evaluate* whether different artifact classes are better served by different backend architectures under the same IR-level semantic interface.

Month 6 Decision Gate Metrics

In Phase I, we will evaluate a small number of representative candidates within each artifact class, rather than a single hand-picked example, and use the following minimum criteria:

- *Extraction/IR viability*: at least two artifact classes must have working extraction pipelines that produce IR fragments with sufficient fidelity for downstream use on a small public corpus.
- *Backend viability*: at least one NSAM-style differentiable backend and one alternative backend must successfully consume the IR while preserving typed interfaces and selected constraints.
- *End-to-end feasibility*: at least two proof-of-concept tasks must run end to end, including one scientific-kernel case and one workflow or program-logic case.
- *Comparative value*: for at least one kernel case, the IR-guided approach must show a meaningful advantage over a baseline at comparable complexity.

If these criteria are not met by Month 6, the project will narrow scope in the remaining Phase I period to the artifact classes and backend mappings that have demonstrated feasibility, while still analyzing failure cases to inform Phase II planning and to clarify where the IR or backend strategy needs revision

Months 6-9: Consolidation, evaluation, and Phase II packaging

- *Evaluate generalizability* across the three artifact classes using common metrics: extraction quality, constraint preservation, interpretability, and task-specific performance.
- *Package* the extraction, IR, backend mappings, and example applications into a reusable prototype with documentation and reproducible examples.
- *Produce a Phase II plan* for scaling from proof-of-concept artifacts to larger DOE codes, workflows, and catalogs, including tighter integration with AmSC, richer formal and differentiable backends, shadow-mode evaluation in realistic software environments, and criteria for selective insertion into production scientific workflows.

Appendix 1: Bibliography & References Cited

1. Nikiema SL, Samhi J, Kaboré AK, Klein J, Bissyandé TF. The code barrier: What LLMs actually understand? arXiv [cs.SE]. 2025. doi:10.48550/arXiv.2504.10557

2. Kravchuk-Kirilyuk A, Graciolli F, Amin N. The modular imperative: Rethinking LLMs for maintainable software. Proceedings of the 1st ACM SIGPLAN International Workshop on Language Models and Programming Languages. New York, NY, USA: ACM; 2025. pp. 106–111.

3. Molins S, Svyatsky D, Xu Z, Coon ET, Moulton JD. A multicomponent reactive transport model for integrated surface‐subsurface hydrology problems. Water Resour Res. 2022;58. doi:10.1029/2022wr032074

4. Velez-Ginorio J, Amin N, Kording K, Zdancewic S. Compiling to recurrent neurons. arXiv [cs.PL]. 2025. doi:10.48550/arXiv.2511.14953

5. Mungall C, Chiba H, Kawashima S, Prins P, Yamamoto Y, Amin N. Logic programming for the biomedical sciences. 2020. doi:10.37044/osf.io/km9ux

6. Mungall CJ, Malik A, Korn DR, Reese JT, O'Boyle NM, Hastings J. Chemical classification program synthesis using generative artificial intelligence. J Cheminform. 2025;17: 152.

7. Arakelyan S, Hakhverdyan A, Allamanis M, Garcia L, Hauser C, Ren X. NS3: Neuro-symbolic semantic Code Search. arXiv [cs.LG]. 2022. doi:10.48550/arXiv.2205.10674

8. Caufield JH, Hegde H, Emonet V, Harris NL, Joachimiak MP, Matentzoglu N, et al. Structured Prompt Interrogation and Recursive Extraction of Semantics (SPIRES): a method for populating knowledge bases using zero-shot learning. Bioinformatics. 2024;40. doi:10.1093/bioinformatics/btae104

9. Tang J, Grant RF, Riley WJ, Brodie E, Graus A, Mekonnen ZA. EcoSIM_a modern ecosystem biogeochemistry model that explicitly represents trait-based plant-microbe-soil interactions. AGUFM. 2024;2024: B41F–1346.

10. Mungall C. BioEPIC data integration around multi-scale simulations. Zenodo; 2023. doi:10.5281/ZENODO.14532950