

A Verified Open Source AI Support Layer of OPALX

From versioned documentation to conversational simulation assistance

Document type: Scientific white paper / discussion draft
Author: A Adelman, acknowledging discussions with the OPALX collaboration & Hugo
Date: 13 August 2026



Abstract

Historically, OPAL—and now OPALX—has been accessed primarily through a command-line interface. While many scientific applications have responded to user demand by introducing graphical interfaces, OPALX has not followed this trend; recent advances in AI now offer a different opportunity: a prompt-first user interface.

OPALX is supported by a large user/developer manual, a physics manual, Doxygen documentation, source code, examples, and extensive unit and regression tests. This white paper asks whether users could interact with that body of knowledge through a single AI prompt: experienced users would obtain current, version-specific answers; new users could describe a problem and receive an input deck or a bounded coarse simulation; and users could upload a failing deck for diagnosis. We conclude that a prompt-first interface is technically credible, but an AI-only replacement of the underlying documentation is neither scientifically defensible nor operationally robust. We propose a versioned retrieval, generation, and validation architecture in which documentation and source remain reviewable evidence, tests become executable knowledge, generated decks are checked in a sandbox, and every answer identifies the OPALX revision and supporting sources. An AI-assisted documentation loop drafts changes when **master** changes while developers retain merge authority. The CSCS LLM Inference API offers a practical sovereign inference plane through OpenAI- and Anthropic-compatible endpoints. Apertus is an attractive transparent and multilingual foundation model, but it should be evaluated against an OPALX-specific benchmark and deployed behind a model-neutral gateway. A staged pilot begins with cited read-only answers, then adds deck conversion and fault diagnosis, and only later permits bounded execution.

Contents

1	Introduction	1
2	Current OPALX knowledge base	1
3	Evidence from related work	1
4	Proposed system architecture	2
4.1	PhysicsCode as an implementation substrate	2
4.2	Versioned ingestion and retrieval	3
4.3	Response contract	4
5	User workflows	4
5.1	Explore OPALX and ask anything	4
5.2	Starting advice and deck synthesis	5
5.3	Conversion from xx.in	5
5.4	Faulty-deck diagnosis	5
5.5	Coarse simulations	5
6	AI-assisted documentation maintenance	5
6.1	Keeping developers in control i.e. in the loop	6
7	CSCS inference service and Apertus	6
7.1	Why the CSCS service is a strong fit	6
7.2	Apertus-first, not Apertus-only	6
8	Conclusion	7

1 Introduction

The strategic question is not whether a language model can produce plausible OPALX text. It is whether an AI service can make support faster while preserving the version identity, physical assumptions, numerical evidence, and developer accountability expected of scientific software.

Goal:: Adopt a *prompt-first, evidence-preserving* support layer. Keep the unified manual, physics material, Doxygen, examples, source, and tests as versioned assets. Let the assistant retrieve, explain, translate, diagnose, execute within policy, and propose documentation changes; do not let it silently redefine OPALX behaviour or merge its own documentation.

Besides this we need to understand that the open source project OPALX has no base funding for computing infrastructure.

The proposal has five user-facing modes:

1. **Explore OPALX:** concept navigation, architecture, elements, algorithms, and version history. Maybe can be viewed as a developer entry.
2. **Convert `xx.in`:** translate a supported foreign or legacy input deck into an OPALX deck, with an explicit mapping report.
3. **Get starting advice:** elicit the physics problem, beam and machine assumptions, accuracy target, and computing constraints before creating a minimal working deck.
4. **Diagnose an uploaded deck:** parse, reproduce the failure in isolation when authorized, explain the cause, and return a patch rather than only prose.
5. **Ask anything:** answer current, cited questions with a visible OPALX commit/release and calibrated uncertainty.

The phrase “replace the documentation” is therefore used only at the interaction layer: a user should not need to know which manual chapter or source directory to search. At the evidence layer, replacement would remove the audit trail required to determine whether a generated answer is correct.

2 Current OPALX knowledge base

OPALX already contains the ingredients of a high-value domain assistant. The challenge is that they have different semantics and update cycles. A manual expresses intended public behavior; Doxygen and comments describe interfaces; source code implements behavior; tests provide executable examples and partial oracles; and issue or support discussions contain useful but non-normative experience. They must not be flattened into an undifferentiated vector store.

Table 1 summarizes the local source snapshot inspected for this paper. Counts are inventory indicators, not quality metrics.

The public repository already validates multiple execution backends and runs unit and selected regression tests in CI [1]. The redesigned manual is text-first and intentionally unifies documentation categories while linking separately generated C++ API pages [2]. This is a favorable starting point: machine-readable sources, code, and tests can be indexed at the same Git revision.

3 Evidence from related work

Retrieval-augmented generation (RAG) was introduced as a way to condition generation on retrieved external evidence rather than parameters alone [3]. Scientific-software projects are now applying this principle to code, documentation, and user workflows. Table 2 highlights the attempts most relevant to OPALX.

Additional studies on Gmsh/Elmer input generation report strong solver-file performance but also expose the need to score completeness, geometry fidelity, and physical outputs rather than syntax alone [11]. Software-engineering benchmarks likewise show that repository tasks require execution environments and test-based evaluation, not answer similarity alone [12].

The Nobile report is particularly informative because it evaluates a local scientific-code agent on IPPL, which is close to OPALX’s ecosystem. Its best reported configuration scored 0.795 ± 0.034 on 100 questions. API usage was strong, while build/install, numerical meaning, and some cross-file reasoning categories were substantially weaker [4]. The report also notes the limitations of one codebase and an LLM judge. For OPALX, this argues for a mixed evaluation: expert review for physics claims and deterministic execution for deck behavior.

Table 1: Indicative OPALX knowledge inventory inspected on 13 August 2026.

Asset	Observed snapshot	Role in the assistant
Unified manual	88 Quarto source files; user, physics, reference, developer, and troubleshooting sections	Normative explanations, formulas, units, workflows, and curated examples
Standalone physics material	46 Quarto source files in a repository now being consolidated into the unified manual	Migration input and specialist physics evidence; duplicates must be detected
C++ implementation	657 .h/.hpp/.cpp/.tpp files	Operational implementation, symbol graph, defaults, exceptions, and version-specific behavior
Unit-test source	84 C++ test files; current local configurations enumerate 76–78 CTest tests	Executable examples and focused behavioral or numerical oracles
Regression tests	External regression repository plus a fast CI subset listed by OPALX	End-to-end behavior and quantitative reference metrics
Doxygen	Repository-level Doxygen configuration and source comments	API surface, call relationships, and developer-facing definitions

4 Proposed system architecture

The system should separate four concerns: versioned knowledge, language-model inference, deterministic tools, and governance. Figure 1 shows the logical design. The inference model may change without changing the evidence contract or validation tools.

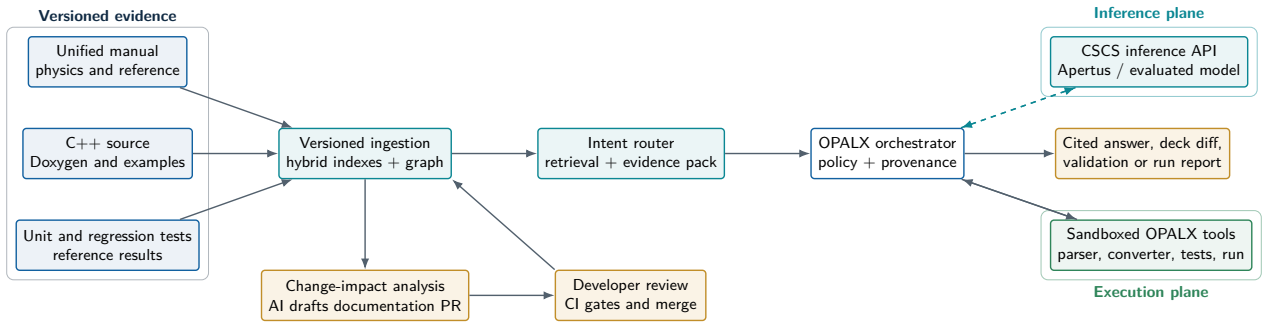


Figure 1: Logical OPALX AI support architecture. Solid arrows carry governed evidence or tool results; dashed arrows carry prompts and model responses. The model never directly merges documentation or invokes unrestricted execution. Developed with the help of Hugo.

4.1 PhysicsCode as an implementation substrate

PhysicsCode is an open-source, MIT-licensed agentic AI coding environment for physics software, simulations, analysis, and research engineering, developed by **PAI Dynamics** [13]. It should not become a new OPALX source of truth; rather, it could supply much of the user-interface, agent-orchestration, model-provider, tool, and retrieval infrastructure in Figure 1. Its public repository already contains a command-line/TUI client, a Tauri desktop application, a VS Code extension, configurable model providers, permission controls, and MCP integration. The provider abstraction accepts custom base URLs, so the CSCS OpenAI-compatible endpoint is a plausible target, subject to an integration test covering authentication, model discovery, streaming, tool calls, timeouts, and error handling.

The most relevant component is **physicscode-science**, a provenance-aware subsystem that is deliberately separated from the interactive agent. It implements a versioned repository registry, exact commit pinning, license detection, content hashing, semantic object extraction, SQLite state, dense/sparse/symbol retrieval, reranking, relationship-aware context, Qdrant-backed production vectors, OpenAI-compatible embedding endpoints, an HTTP/MCP service, readiness checks, and retrieval and agentic evaluation commands. OPALX

Table 2: Related attempts and the design lesson for OPALX.

System	Scope	Relevant result or method	OPALX lesson
Nobile scientific-code agent [4]	Local RAG over the IPPL C++ codebase	Offline structural ingestion, explanations, vector retrieval, reranking, and a 100-question evaluation	Reuse the benchmark discipline and structured retrieval; add executable validation and human scoring
PETSc assistants [5]	Source, manuals, issues, discussions, papers	RAG, reranking, chat, developer support, and proposed updates to formal documentation	Treat AI as a bridge into a heterogeneous knowledge base and as a documentation contributor
CelloAI [6]	HEP/HPC documentation and generation	Syntax-aware chunking, call graphs, Doxygen generation, local deployment	Preserve C++ structure and dependency context; local or sovereign operation is feasible
AstraAI [7]	AMReX code generation and editing	Combines semantic retrieval with AST-derived constraints and reviewable edits	Retrieval alone is insufficient for structured artifacts; use a parser and an intermediate representation
RHIC assistant [8]	Long-term access to experimental knowledge and software	Provenance-rich multi-format ingestion, access control, RAG, and composable tools	Attach source, timestamp, revision, and access label to every knowledge unit
Pasimodo RAG [9]	Closed-source multibody simulation models	Local RAG helps model creation, but failures often trace to missing or poorly selected information	Diagnose retrieval insufficiency explicitly; curate task-specific evidence bundles
GENIUS [10]	Quantum ESPRESSO input generation and execution	Knowledge graph plus a finite-state recovery loop; about 80% of 295 prompts passed early execution validation	Input generation becomes credible only when a deterministic checker and recovery loop close the system

is already present in its public repository registry as an enabled, high-priority source on `master`. This is useful initial integration, but it does not yet establish that the production index is complete or current: the OPALX user manual and physics manual are not registered, and no OPALX-specific benchmark result is published. In a local inspection of the public snapshot on 31 August 2026, all 61 `physicscode-science` unit tests passed; this indicates an implemented substrate, not validated OPALX answer quality.

Using PhysicsCode for a service such as `opalx.ai.psi.ch` would require the following OPALX-specific work:

1. **Complete and align the corpus.** Register the OPALX user and physics manuals, generated Doxygen, examples, tests, and reference results. Pin all artifacts to a common OPALX commit and refresh them from CI when `master` changes.
2. **Implement OPALX tools.** Expose deterministic MCP tools for input-deck parsing, semantic validation, conversion, deck comparison, executable discovery, controlled OPALX runs, and result/log collection. Each run must record the commit, build configuration, MPI rank count, backend, resource limits, and exit status.
3. **Enforce the response contract.** Convert retrieved provenance into claim-level citations and make the revision, assumptions, validation state, and generated diffs visible in the UI. Retrieval confidence alone must never be presented as physics validation.
4. **Harden the hosted service.** Package the web experience for `opalx.ai.psi.ch`; add PSI authentication, authorization, quotas, audit records, secret management, retention policy, and isolated execution. Configure and test the CSCS inference provider while retaining a model-neutral gateway.
5. **Create acceptance benchmarks.** Add expert-reviewed OPALX questions, input-deck synthesis and conversion cases, faulty decks, build failures, and representative single- and multi-rank runs. Connect these gates to both PhysicsCode releases and OPALX documentation/index updates.
6. **Close the developer loop.** Implement change-impact analysis and reviewable documentation pull requests, but leave approval, CI policy, and merge authority with OPALX developers.

Table 3 gives a preliminary, weighted engineering estimate of reusable implementation. On this basis, approximately **50% of the total OPALX support-service engineering scope is already present**, including roughly three quarters of the generic agent and retrieval substrate. Most of the remaining work is OPALX-specific integration, validation, security, and operations. This percentage is a component-coverage estimate, not a measured 50% task-success rate; it should be replaced by benchmark evidence during the pilot.

4.2 Versioned ingestion and retrieval

Every indexed knowledge unit should carry at least:

Table 3: Preliminary PhysicsCode reuse estimate for the proposed OPALX service. Weights represent the engineering scope assumed in this white paper.

Capability area	Weight	Present	Weighted points
Agent interfaces and model-provider abstraction	15%	80%	12.0
MCP, tools, workspace, and permission substrate	15%	70%	10.5
Scientific ingestion, retrieval, provenance, and evaluation framework	25%	80%	20.0
Complete OPALX corpus integration and continuous synchronization	15%	20%	3.0
OPALX deck validation and controlled execution tools	15%	10%	1.5
Hosted-service security, identity, quotas, and operations	10%	20%	2.0
OPALX benchmark, governance, and documentation loop	5%	20%	1.0
Total	100%		50.0

- repository, branch/tag, commit hash, build configuration, source path, line or section identity, and ingestion timestamp;
- content class (manual, formula, API, source, test, regression metric, example, issue) and authority level;
- OPALX object, command, attribute, symbol, units, applicable tracking mode, backend, and dependency relationships where available;
- review state, last human modifier, access classification, and supersession/duplication links.

Retrieval should be hybrid. Dense embeddings find conceptual matches; lexical search preserves exact OPALX keywords and error strings; an AST/symbol graph connects declarations, implementations, commands, tests, and Doxygen; and a domain graph connects physical concepts, variables, units, elements, and reference examples. Query routing then composes a small evidence pack appropriate to the task.

A useful formal acceptance rule is

$$\text{accept}(y) = C(y, K_c) \wedge V(y, T_c) \wedge P(y, u), \quad (1)$$

where y is the proposed answer or artifact, K_c is the knowledge snapshot at commit c , C requires claim-level evidence, T_c is the set of applicable deterministic checks, V requires those checks to pass, and P enforces user and execution policy. A response may still be useful when V is unavailable, but it must be labelled “unverified” rather than silently accepted.

4.3 Response contract

Every answer should include a machine-readable envelope even when the UI renders friendly prose:

- OPALX release/commit and knowledge-index version;
- direct answer, assumptions, and uncertainty;
- citations to the manual, source, tests, or examples supporting each material claim;
- generated artifacts and a semantic diff from the upload or source deck;
- checks executed, executable version, rank/backend, resource limits, exit status, and retained logs;
- an escalation path when evidence conflicts or the requested regime is outside validated coverage.

This contract turns “up to date” into a testable statement: an answer is current relative to a named snapshot, not merely generated today.

5 User workflows

5.1 Explore OPALX and ask anything

Read-only questions are the safest first release. The assistant should ask whether the user targets a release or **master**, retrieve the minimum evidence required, and link to the exact relevant material. Experienced users may request terse symbol- and commit-level answers; new users may request a conceptual explanation with a minimal example. The underlying evidence is identical, while response depth changes.

For physics questions, the answer template should require definitions, equations where useful, units, coordinate system, model assumptions, boundary conditions, numerical method, and known validity range. If any item is

absent from maintained evidence, the assistant must say so. Source-code behavior alone is not enough to infer a physical guarantee.

5.2 Starting advice and deck synthesis

The assistant should first turn an underspecified scientific request into a reviewable problem specification:

1. beam species, charge convention, reference energy/momentum, distribution, and intensity;
2. lattice/elements, coordinate conventions, fields, collective effects, and apertures;
3. requested observables, fidelity, random seeds, stopping conditions, and acceptable runtime;
4. target OPALX revision, execution backend, MPI ranks/threads/accelerators, and available resources.

Only then should it generate a deck. The LLM should produce a typed intermediate representation rather than free-form final syntax. A deterministic emitter serializes that representation into canonical OPALX syntax. The response contains both the deck and a table mapping each requested physical concept to the OPALX construct, source citation, value, and unit.

5.3 Conversion from `xx.in`

Conversion is a compiler problem assisted by an LLM, not a translation prompt. A source-format parser creates a source intermediate representation; deterministic rules translate well-defined constructs; the LLM proposes mappings only for ambiguous or high-level intent; and an OPALX emitter generates the target deck. Unsupported semantics are returned as structured warnings. Round-trip or invariant checks should compare lattice length, reference trajectory, energy profile, integrated fields, particle species, distribution moments, and other code-pair-specific quantities where meaningful.

The converter must distinguish three states: *equivalent*, *approximated with a stated model difference*, and *not translated*. A syntactically valid deck is never called physically equivalent without quantitative evidence.

5.4 Faulty-deck diagnosis

An uploaded deck is both untrusted data and a potentially valuable reproducer. The service should:

1. store it in a quarantined workspace, remove secrets and unsafe paths, and inventory included files;
2. parse its grammar and dependencies without granting arbitrary shell or network access;
3. retrieve exact error strings, command definitions, examples, and recent code changes at the target revision;
4. run only an approved minimal reproduction with CPU/time/memory/output limits when the user authorizes execution;
5. return the root cause, a minimal patch, citations, executed checks, and remaining uncertainty.

The inspected OPALX command-line path parses and executes statements together and does not expose an obvious pure `-check` mode. A non-executing validation API or restricted parser front-end is therefore a recommended enabling feature before accepting arbitrary uploads. Until then, parsing itself must occur inside the same sandbox as execution.

5.5 Coarse simulations

“Run a coarse simulation” must be an explicit, costed action rather than an implicit chatbot side effect. The assistant proposes a reduced problem, explains every fidelity reduction, estimates resources, and waits for user confirmation. The execution service then submits a controlled job, streams bounded status, and returns a run manifest plus selected diagnostics. The result must state what was changed: particle count, mesh, time step, solver tolerance, enabled physics, ranks/threads, seed, and stopping criteria. Coarse output supports setup and debugging; it does not replace a production convergence study.

6 AI-assisted documentation maintenance

The documentation-maintenance loop should start from each accepted change to `master` or from a release candidate:

1. compute the code/API/test diff and map changed symbols, commands, defaults, units, exceptions, and regression expectations to manual sections;
2. run existing documentation link/build checks and a stale-claim detector;
3. have the model draft a small documentation pull request with source citations and an impact statement;

4. assign domain owners based on the affected physics or subsystem;
5. require human review plus manual build, unit tests, and relevant regression tests before merge;
6. rebuild the index only from the reviewed merged snapshot, retain the previous index, and publish the new commit identity.

The AI may also draft missing Doxygen and test explanations, but generated comments must not merely restate code. For physics or numerics, a proposed change must identify equations, units, boundary conditions, discretization, validity assumptions, and expected effects on stability, conservation, reproducibility, floating-point order, and parallel execution where applicable.

6.1 Keeping developers in control i.e. in the loop

Developer control should be visible in the repository workflow:

- the bot opens a labelled draft pull request and never approves or merges it;
- each paragraph is traceable to changed code, a test, an issue, or an existing authoritative section;
- disputed claims are converted to review questions rather than resolved by the model;
- accepted and rejected suggestions become evaluation data, not private fine-tuning data by default;
- emergency rollback selects a previous immutable knowledge index without reverting OPALX source.

This approach can reduce the cost of maintenance while preserving authorship, technical ownership, and review history. The long-term measure of success is not the volume of generated documentation; it is lower documentation latency with no increase in defect escape.

7 CSCS inference service and Apertus

7.1 Why the CSCS service is a strong fit

The CSCS LLM Inference API provides managed, OpenAI- and Anthropic-compatible endpoints for a curated set of open-weight models. CSCS states that prompts and responses are not recorded and remain on infrastructure it controls in Switzerland. The service exposes model discovery, chat completion, messages, and embedding endpoints; access is project-level and backed by inference credits [14]. These properties align well with OPALX’s academic setting and the desire for sovereign handling of unpublished input decks.

The API is maybe first an inference plane, not the complete OPALX service.

The service intentionally offers a reduced, centrally updated model set and does not support private model deployment [14]. Therefore the OPALX gateway should call `/v1/models` at deployment/health-check time, map logical roles to an allowlisted model, and record the exact returned model identifier with every answer. A domain-adapted private model would require a separately operated serving path, for example on Alps, rather than an assumption that the managed service will host it.

7.2 Apertus-first, not Apertus-only

Apertus is a compelling foundation for the project. The original 8B and 70B releases are fully open, multilingual, support contexts up to 65,536 tokens, and publish training, data, and evaluation details [15]. CSCS has made Apertus available through its inference service and is extending the model family [16]. This transparency is valuable for governance and reproducibility, while multilingual support can serve an international accelerator community.

However, OPALX requires specialized C++, input-language, accelerator-physics, and debugging capability. Apertus’ published general coding benchmarks show useful capability but do not establish OPALX competence, and several code-oriented open-weight comparators score higher on those generic tasks [15]. The correct decision is empirical:

- evaluate the currently served Apertus 8B/70B and new Apertus releases on the same frozen OPALX benchmark;
- use a smaller model for intent routing or summarization only if it meets evidence and latency gates;
- use the best validated model for each high-risk role, including a code-specialized model if Apertus does not meet deck-generation or repair thresholds;
- retain an Apertus-only profile for experiments where full openness and Swiss sovereignty outweigh maximum task performance.

This preserves the strategic value of Apertus without making physical correctness depend on brand or

parameter count.

8 Conclusion

A single AI prompt can plausibly become the best front door to OPALX. It can answer current questions, teach new users, translate supported input formats, diagnose failures, and eventually coordinate bounded coarse simulations. Comparable efforts in PETSc, HEP software, AMReX, RHIC knowledge preservation, and materials simulation show that the direction is realistic. They also show that dependable systems combine language models with structured retrieval, parsers, knowledge graphs, execution, and evaluation.

The manuals, Doxygen, source, examples, and tests should therefore be consolidated as a versioned evidence system, not discarded. The AI maintains and navigates that evidence, while developers approve changes and executable checks constrain generated artifacts. CSCS offers a strong sovereign inference foundation, and Apertus should be the first transparent Swiss model evaluated. The architecture should nevertheless remain model-neutral because OPALX competence must be measured on OPALX tasks.

References

- [1] OPALX Project. OPALX: Object oriented parallel accelerator library for exascale. Source-code repository, 2026. URL <https://github.com/OPALX-project/OPALX>. Accessed 13 August 2026.
- [2] OPALX Project. OPALX documentation: Unified user, physics, reference, developer, and troubleshooting manual. Quarto source repository, 2026. URL <https://github.com/OPALX-project/opalx-manual>. Accessed 13 August 2026.
- [3] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktaschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>.
- [4] Aaron Nobile. Retrieval-augmented generation for scientific code understanding. Semester project report, ETH Zurich, Department of Computer Science / Physics, July 2026. Supervised by Andreas Adelmann and Mohsen Sadr; implementation available at <https://github.com/nobileaaron/rag-scientific-code-agent>.
- [5] Barry Smith, Junchao Zhang, Hong Zhang, Lois Curfman McInnes, Murat Keceli, Archit Vasan, Satish Balay, Toby Isaac, Le Chen, and Venkatram Vishwanath. AI assistants to enhance and exploit the PETSc knowledge base. *arXiv preprint arXiv:2506.20608*, 2025. doi: 10.48550/arXiv.2506.20608. URL <https://arxiv.org/abs/2506.20608>.
- [6] Mohammad Atif, Kriti Chopra, Ozgur Kilic, Tianle Wang, Zhihua Dong, Charles Leggett, Meifeng Lin, Paolo Calafiura, and Salman Habib. CelloAI: Leveraging large language models for HPC software development in high energy physics. *arXiv preprint arXiv:2508.16713*, 2025. doi: 10.48550/arXiv.2508.16713. URL <https://arxiv.org/abs/2508.16713>.
- [7] Mahesh Natarajan, Xiaoye Li, and Weiqun Zhang. AstraAI: LLMs, retrieval, and AST-guided assistance for HPC codebases. *arXiv preprint arXiv:2603.27423*, 2026. doi: 10.48550/arXiv.2603.27423. URL <https://arxiv.org/abs/2603.27423>.
- [8] Mohammad Atif, Vincent Garonne, Eric Lancon, Jerome Lauret, Alexandr Prozorov, and Michal Vranovsky. AI-powered assistant for long-term access to RHIC knowledge. *arXiv preprint arXiv:2509.09688*, 2025. doi: 10.48550/arXiv.2509.09688. URL <https://arxiv.org/abs/2509.09688>.
- [9] Andreas Baumann and Peter Eberhard. Experiments with large language models on retrieval-augmented generation for closed-source simulation software. *Multibody System Dynamics*, 2026. doi: 10.1007/s11044-026-10164-7. URL <https://doi.org/10.1007/s11044-026-10164-7>.
- [10] Mohammad Soleymanibrojeni, Roland Aydin, Diego Guedes-Sobrinho, Alexandre C. Dias, Mauricio J. Piotrowski, Wolfgang Wenzel, and Celso Ricardo Caldeira Rego. GENIUS: An agentic AI framework for autonomous design and execution of simulation protocols. *Communications Materials*, 7(115), 2026. doi: 10.1038/s43246-026-01167-0. URL <https://doi.org/10.1038/s43246-026-01167-0>.
- [11] Ossama Shafiq, Amin Rahmat, Alessio Alexiadis, Bahman Ghiassi, and Andrea Serani. Evaluating the performance of large language models for geometry and simulation file generation in physics-based simulations. *Applied Sciences*, 15(22):12114, 2025. doi: 10.3390/app152212114. URL <https://doi.org/10.3390/app152212114>.

- [12] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.
- [13] PAI Dynamics. Physicscode: An open-source agentic AI coding environment for physics. Open-source software repository, 2026. URL <https://github.com/PAIdynamics/code>. MIT licensed; accessed 31 August 2026.
- [14] Swiss National Supercomputing Centre. LLM inference API service. Technical documentation, 2026. URL <https://docs.cscs.ch/services/inference/api/>. Accessed 13 August 2026.
- [15] Alejandro Hernandez-Cano et al. Apertus: Democratizing open and compliant LLMs for global language environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 46877–46955. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.2172. URL <https://aclanthology.org/2026.acl-long.2172/>.
- [16] Swiss National Supercomputing Centre. Apertus 1.5: Building the next generation of open AI infrastructure. CSCS article, 2026. URL <https://www.cscs.ch/science/computer-science-hpc/2026/apertus-15-buildin-g-the-next-generation-of-open-ai-infrastructure>. Accessed 13 August 2026.