STATUS OF OSPREY: A FRAMEWORK FOR AGENTIC AI IN CONTROL SYSTEMS

T. Hellert*[1], G. Martino[1], R. Lehe[1], A. Huebl[1], K. Iliev[1], S. C. Leemann[1], A. Sulc[1], J.-L. Vay[1], A. Wu[1], E. Zoni[1], Z. Zhang[2], N. Kuklev[3], M. Smith[4], H. Shang[4], C. Xu[4], A. Carpenter[5], N. Wang[6], and the Multi-Office Accelerator Team
[1]Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[2]SLAC National Accelerator Laboratory, Menlo Park, CA, USA
[3]Fermi National Accelerator Laboratory, Batavia, IL, USA
[4]Argonne National Laboratory, Lemont, IL, USA
[5]Thomas Jefferson National Accelerator Facility, Newport News, VA, USA
[6]Cornell University, Ithaca, NY, USA

Abstract

Operating large-scale scientific facilities requires coordinating diverse subsystems, translating operator intent into precise hardware actions, and maintaining strict safety oversight. Language-model agents offer a natural interface for these tasks, but most existing approaches are not yet reliable or safe enough for production use. We introduce Osprey, a framework that wraps a coding agent in a control-room operator interface, a tool surface that reaches hardware through pluggable connectors for the control system used in our community, and a first-class component for natural-language search of facility electronic logbooks. The agent itself is treated as a replaceable component: operator interface, safety policy, tool servers, and connectors stay under facility control, while the agent backend can be swapped as the AI ecosystem evolves. A declarative build-profile mechanism lets each facility maintain its own configuration without forking the shared framework, keeping deployments reproducible across updates. Osprey has been deployed at several DOE accelerator facilities through the MOAT seed effort within the Genesis Mission. This paper presents the current framework architecture and reports on the substantial evolution Osprey has undergone over the past year.

INTRODUCTION

Operating an accelerator means coordinating many heterogeneous subsystems, translating operator intent into precise hardware actions, and maintaining strict safety oversight. Today, that work depends on graphical control interfaces and on tacit expert knowledge held by a small team. Reasoning across subsystems, finding the right control variable, recalling a past procedure, or assembling context during a fault still falls largely on individual operators.

Language-model coding agents have matured rapidly outside our community and are an attractive interface for this kind of work. Commercial agent products, however, do not come with the connectors, safety policy, facility knowledge, or operator workflows required for a control-room deployment. Closing that gap in a way that is reproducible across various facilities is the architectural problem this paper addresses.

Osprey's prior incarnation [1] was a self-maintained orchestrator with task extraction, capability classification, and planning implemented in-house on top of a fixed agent backend. A recent DOE workshop on agentic AI for user facilities [3] since recommended the opposite approach: domain-specific integration layers built atop general-purpose agent frameworks rather than competing ones, and a "co-pilot" pattern with default read-only access and human-approved writes. The architecture reported here is a substantial evolution in line with both findings — the in-house orchestrator has been retired in favor of a general-purpose coding agent, the periphery (operator interface, safety chain, tool servers, connectors) has been redesigned to stay under facility control while the agent itself becomes a replaceable component, and a declarative build profile lets each facility deploy and re-deploy its own configuration reproducibly. The framework has been deployed at seven DOE accelerator facilities under the MOAT seed effort within the Genesis Mission [4], with the Ariel logbook component presented separately in MOP6326 [5].

ARCHITECTURE

Figure 1 shows the resulting layout: a stable periphery, comprising the operator interface, the safety chain, the MCP tool surface, and the hardware connectors, wrapped around an externally-sourced coding agent, with the subsections below taking each layer in turn.

External Coding Agent

The coding agent at the center of Osprey — the software that turns a language model into a working assistant with tool access, conversation memory, and execution control — is the part of the AI stack moving fastest, with commercial and open-source projects releasing significant changes on a timescale of weeks. A small facility team cannot keep pace by maintaining its own, so Osprey treats this layer as an external dependency, swapped when the surrounding ecosystem makes a different choice attractive. The current reference is Claude Code, with the language model behind it

* thellert@lbl.gov

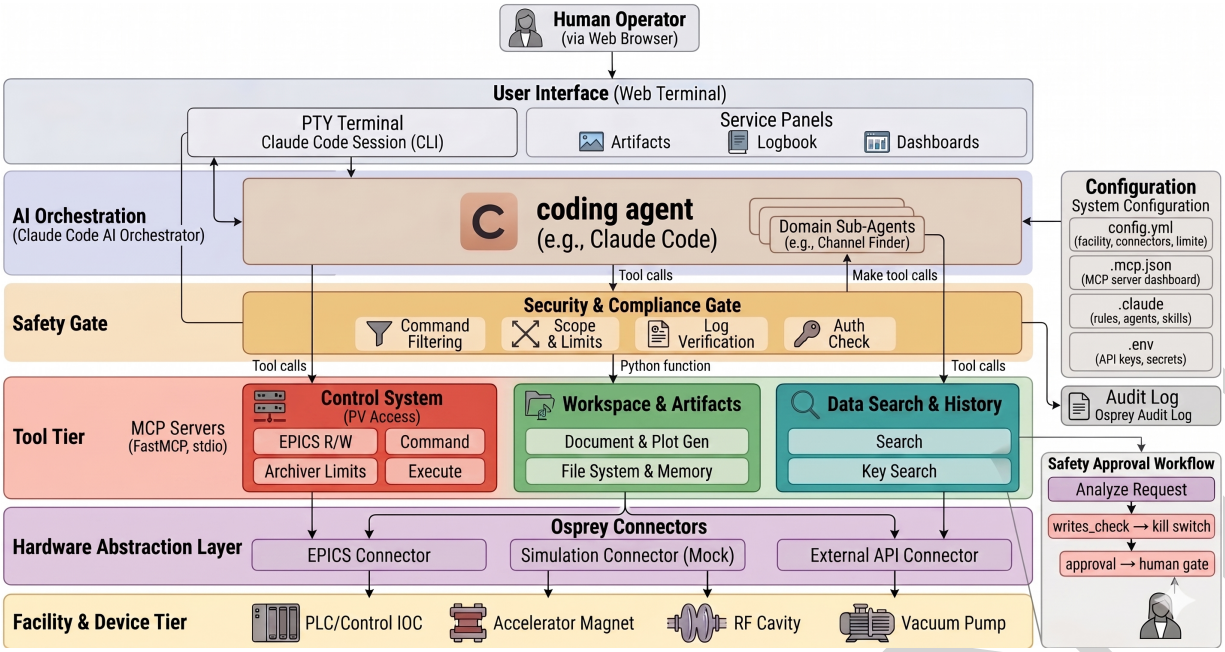

Figure 1: Osprey system architecture: from operator to facility, with the safety gate and approval workflow in-line. The coding agent is the only externally-sourced layer; everything above and below is owned by Osprey.

selectable from a single configuration field across more than ten providers, and alternatives are under active evaluation.

Collaborative Operator Interface

The interface is a web-based workbench where operator and agent share a screen, designed for genuine collaboration rather than as a chat box reused from a generic product. Configuration of an otherwise complex system is kept accessible: a settings tab, for instance, lets operators adjust agent behavior without touching code. Each facility composes the workbench from a common library of panels. A default panel browses the artifacts produced during an agent session (interactive plots, tables, lattice visualizations, captured screens, and auto-generated notebooks), and additional domain panels can be enabled per deployment, including logbook search, tuning, channel-finder browsing, and lattice dashboards.

Safety Chain

A request begins as natural language and reaches the machine only through a chain of independent checks. The language model itself produces only text: it cannot execute commands, and the code it proposes is first screened by static analysis for risky patterns so that obviously unsafe code is blocked before a human reviews it. A human operator then inspects the proposed action and chooses to approve, modify, or reject it, so that no machine-touching command proceeds without explicit consent. Once approved, the action is checked against a whitelist of permitted channels and against per-channel limits drawn from the facility's safety database, and unlisted channels or out-of-range values are refused. Only then does the validated command execute, inside a containerized environment with a controlled network interface to the control system, so that the agent never gains direct access to the production network. The chain is specified as a portable pattern: no single misconfiguration or implementation choice can grant unmediated access to the machine.

Tools, Subagents, and Connectors

The agent reaches the machine through a modular stack of plug-in components: MCP (Model Context Protocol) tool servers that expose facility operations, subagents that take on focused subtasks in their own context windows so the main conversation stays lean, skills that package workflow-specific instructions, and connectors that abstract over the underlying control protocol. The current stack covers control-system access, channel finding, and logbook search, with more capabilities in active development. Because each capability is itself a small bundle of these component types, a facility can ship a complete extension rely on the local connector implementation to bind it to the on-site control system.

Control-system access is presented through a single MCP server that exposes read, write, archiver, and limit-checking operations, with an internal connector layer that abstracts over the underlying protocol. In-tree connectors target EPICS, LabVIEW, and DOOCS, so the same agent code runs against fundamentally different control systems without modification.

The channel finder [6,7] translates natural-language intent into control-system addresses, and Osprey invokes it through a dedicated subagent to keep the resolution traffic isolated from the main operator-facing context. Several resolution strategies are available, so the approach can be matched to the size and naming convention of each facility's namespace.

The Ariel logbook search [5] gives the agent access to operational history through pluggable ingestion adapters that normalize entries from different in-house logbook systems, with the search step likewise wrapped in a subagent.

Facility Development

Each facility maintains its own profile, a small declarative artifact layered over the shared Osprey platform. The platform evolves on its own release cadence, and a facility upgrades by pointing its profile at a newer platform version. Facility-specific customization and platform improvement therefore do not compete for the same engineering surface, and a new facility coming online or an existing one tracking a platform release is a routine operation rather than a per-site engineering project.

Beyond configuration, a profile can ship facility-specific extensions: custom operator skills, subagents tuned to local workflows, additional MCP servers, and deployment-time scripts. The platform itself stays minimal. Everything specific to a facility lives in its profile, so two facilities can customize independently without forking the shared code.

As a canonical example, the ALS maintains its profile repository alongside its custom skills and operator workflows [8], with a continuous-integration pipeline that builds control-room containers on each profile release. Multiple containers are deployed in parallel with different access rights so that a single Osprey installation can serve distinct user contexts without compromising the safety boundary, and this per-context scoping is itself part of the build profile and therefore portable to other facilities. The control room pulls the appropriate container on demand, decoupling platform upgrades from operational rollout. Multi-user authentication and a planned move to Kubernetes-based orchestration are on the near-term roadmap.

MULTI-FACILITY ADOPTION

The same Osprey codebase has been deployed at seven DOE accelerator facilities under the Genesis Mission MOAT-seed effort.

A single natural-language prompt drove a full insertion-device hysteresis measurement at the ALS [2]: the agent identified the relevant process variables, retrieved archival baselines, generated a bidirectional scan script, and executed it under operator approval through containerized EPICS gateways, returning a publication-quality hysteresis plot. Preparation effort dropped by approximately two orders of magnitude relative to expert manual scripting, and every write passed through the safety chain.

At LCLS, a natural-language tune-up request led Osprey to assemble and deploy a complete Xopt/Badger optimization routine, drawing algorithm choice, variables, ranges, and objectives from the performance history of prior runs, and substantially improved FEL pulse intensity in live operation. This addresses a long-standing barrier to routine use of such tools, where configuring a single run by hand otherwise takes operators up to 15 minutes.

In Main Accelerator Complex troubleshooting at Fermilab, Osprey compared electronic logbook entries, parameter-database metadata, and live ACNET readings, and identified within minutes that a retrieved historical explanation was inconsistent with the actual machine configuration. Fermilab accelerator experts subsequently confirmed both the inconsistency and Osprey's interpretation of the live machine state.

Across these three deployments the same architecture serves very different machines, control systems, and operator workflows, with only the per-facility profile differing between sites. The lesson from a year of deployment is to maintain the periphery, not the coding agent itself: framework-portable patterns survive the rapid churn of the AI ecosystem. The framework is openly available [9].

ACKNOWLEDGEMENTS

This work was supported by the Advanced Light Source and by the DOE Genesis Mission MOAT seed effort (DE-FOA-0003612).

REFERENCES

[1] T. Hellert *et al.*, "Osprey: an agentic AI framework for particle accelerator operations," APL Mach. Learn., Feb 2026. doi:10.1063/5.0306302

[2] T. Hellert, D. Bertwistle, S. C. Leemann, A. Sulc, and M. Venturini, "Agentic artificial intelligence for multistage physics experiments at a large-scale user facility particle accelerator," *Phys. Rev. Res.,* vol. 8, p. L012017, Jan 2026. doi:10.1103/jtqy-9jz1

[3] T. Hellert *et al.*, "Agentic AI for User Facilities: Workshop Report," Synchrotron Radiation News, 1-15. doi:10.1080/08940886.2026.2649097

[4] J.-L. Vay *et al.*, "Overview of the US DOE Multi-Office Particle Accelerator Team (MOAT) Project," presented at IPAC'26, Deauville, France, May 2026, paper MOV6302, this conference.

[5] T. Hellert *et al.*, "ARIEL: Agentic Retrieval Interface for Electronic Logbooks," presented at IPAC'26, Deauville, France, May 2026, paper MOP6326, this conference.

[6] T. Hellert *et al.*, "From Natural Language to Control Signals: A Conceptual Framework for Semantic Channel Finding in Complex Experimental Infrastructure." doi:10.48550/arXiv.2512.18779

[7] T. Hellert, "Bridging the Gap Between Control Systems and Natural Language: A Framework for Semantic Channel Finding," presented at IPAC'26, Deauville, France, May 2026, paper FRO6M01, this conference.

[8] ALS facility profiles, `https://git.als.lbl.gov/physics/production/als-profiles`, 2026.

[9] ALS-APG, "Osprey," `https://als-apg.github.io/osprey`, 2026.