

AUTONOMOUS PLANNING AND EXECUTION OF INJECTOR TUNING VIA THE OSPREY AGENTIC FRAMEWORK AT THE ALS

G. Martino*, T. Hellert, S. C. Leemann, A. Sulc

Lawrence Berkeley National Laboratory, Berkeley, CA, United States

Abstract

We take a step beyond operator-supervised optimization by introducing an agent that plans and executes fast, sample-efficient injector tuning constrained by competing goals (capture efficiency, energy spread, transmitted charge). We present an agent built on the Osprey agentic framework that can (i) author a machine-readable configuration for a configurable online optimizer, (ii) invoke that optimizer as a tool, and (iii) autonomously infer next tuning actions by querying and actuating the control system. The agent reasons over multi-diagnostic signals and operational constraints, selects objective/constraint formulations, proposes bounded setpoint updates, and schedules measurements; safety is enforced via capability-scoped tool use, guardrails, and machine-protection checks. Control system integration relies on EPICS primitives for live state estimation and closed-loop execution. Initial machine studies at the ALS show the agent can compose full tuning plans end to end: assessing live machine state, selecting a procedure, configuring the optimizer, and invoking runs through capability-scoped tool calls. This reduces operator intervention from per-knob supervision to high-level approval at orchestration checkpoints. We summarize the architecture, online decision logic, early results, and a path to portable deployment at other accelerator facilities.

INTRODUCTION

The Advanced Light Source (ALS) [1] at Lawrence Berkeley National Laboratory is a DOE user facility, and uninterrupted user delivery requires the injector to feed the booster reliably during topoff operation. Maintaining high performance day to day requires regular tuning across the injector's RF, steering, and timing systems. Today, this tuning is performed primarily by operators. The process is time-consuming, its results can vary from shift to shift, and it draws on the experience of a small group of experts. Machine learning has made the inner loop of accelerator tuning increasingly tractable [2], and recent LLM-driven prototypes have shown the feasibility of natural-language interfaces to real-machine control [3, 4].

A companion paper [5] introduces *tuning_scripts*, an online-optimization framework that automates individual tuning runs by orchestrating a pool of online optimizers under a layered safety envelope. Its scope is the inside of a single optimization run. The orchestration around runs remains operator-driven: assessing the machine state, picking which procedure to run, generating the optimizer configuration, interpreting the run summary, and deciding the next

move. These inter-run steps are a source of shift-to-shift variability that the per-run optimizer cannot close on its own. Handing that orchestration to a free-running LLM is not a viable alternative either: unconstrained tool use against a live accelerator is unsafe in ways the existing protection systems were not designed to catch.

Our goal is to reduce operator intervention from per-knob supervision to high-level approval, with safety enforced by construction rather than by convention. We pursue it through three pieces. First, the Osprey agentic framework exposes every agent capability as a typed, capability-scoped tool with a mandatory human-in-the-loop (HITL) gate. Second, an MCP wrapper exposes *tuning_scripts* as one tool among many, so the agent reasons about the optimizer alongside diagnostics and reference docs rather than treating it as a black box; actuation stays inside the optimizer's bounded run, not in the agent. Third, every run terminates at an operator decision: commit the optimizer's best point or revert to the starting setpoints. No agent-driven change becomes permanent without explicit operator approval.

OSPREY AGENTIC FRAMEWORK

Osprey [6–8] is a framework for deploying LLM agents in safety-critical accelerator control systems, deployed at the ALS and open-source¹. Three of its properties matter directly here: capability-scoped tools whose action space is bounded at registration time rather than at runtime; tool servers built on the Model Context Protocol (MCP) [9], giving a uniform interface for control-system primitives (EPICS *get/monitor*) and higher-level endpoints (optimizer, reference-doc retrieval); and a mandatory HITL gate at every actuation, with a fail-safe halt-and-explain default on denied approval. The full safety architecture is described in those references; we reuse it without modification.

MCP Integration with *Tuning_scripts*

Figure 1 shows the agent–tools–machine layout. The agent's primary tool is *tuning_scripts* [5], a FastAPI service that exposes a pool of online optimizers (Bayesian, RCDS, extremum seeking, NSGA-II, ...) and two procedures the agent draws on: a *cold-start bring-up* that walks the injector from a zeroed state to beam at the booster entry through stage-gated charge monitors, and a *close-bounds optimization* that refines around a known working point with bounds shrunk to ~10% of the safe range. A plugin layer selects the hardware backend (live ALS, ALS read-only, simulator) without changing the optimization core.

* gmartino@lbl.gov

¹ <https://github.com/als-apg/osprey>

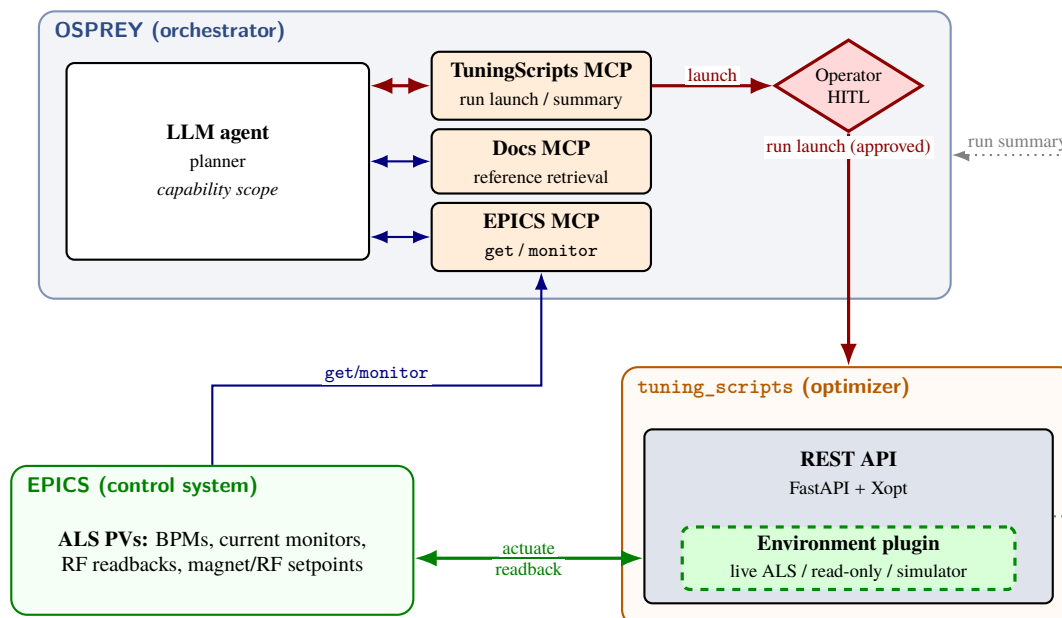


Figure 1: Three-zone agent architecture: **Osprey** orchestrator (top) hosting the LLM agent and its MCP tool servers, **EPICS** control system (bottom-left), and **tuning_scripts** optimizer (bottom-right).

TuningScriptsMCP bridges the two: it is an MCP server that drives the `tuning_scripts` REST API on the agent’s behalf, exposing two classes of tool: `run launch`, parameterized by a JSON configuration that specifies objective, knobs, bounds, step limits, and surrogate type; and structured run summaries that feed back into the agent’s reasoning context after a run completes. Run launch is the actuation step of the loop, so it inherits Osprey’s HITL guarantee: the agent submits a configuration, the operator reviews it at the gate shown in Fig. 1, and `tuning_scripts` only sees the run after explicit approval; a denial halts the agent in the fail-safe halt-and-explain default. Capability scope restricts which environments the agent may target according to the privilege granted to a given session: simulator, ALS read-only mirror, or live ALS. The choice of optimizer and procedure remains the agent’s own.

The agent never writes to knobs. Actuation happens inside the bounded optimizer run that `tuning_scripts` owns, while live machine state flows back into the agent through EPICS `get` and `monitor`. Because the same `tuning_scripts` REST API backs the Dash web UI used by operators, agent-driven runs surface live in that dashboard in the same form as manually-driven runs, observable end to end without agent-specific instrumentation.

AGENT ORCHESTRATION PROCEDURE

The agent’s central responsibility is plan composition: from an assessment of the live machine state, decide what to optimize, which knobs to tune, with what algorithm and within what bounds, then author a JSON configuration that the optimizer can execute. A single tuning episode is one or more such plans, chained based on what the previous run produced.

`tuning_scripts` exposes an environment with a fixed menu but a free choice on it: a defined set of objective signals (e.g., charge monitors along the injector chain) and a defined set of variables (RF phases, gun trigger timing, focusing/steering magnets) with their safe operating bounds. The agent composes a plan by selecting a subset of these, an algorithm (Bayesian optimization, RCDS, extremum seeking, ...), and bounds that may be narrowed around the current setpoint. Predefined procedures from [5] sit as starting templates to specialize or recombine, not as a fixed menu to select from.

State signals drive the composition. Charge readbacks at the gun output, post-linac, and booster entry tell the agent whether the injector is fully zeroed, partially tuned, or already delivering beam, while the operator’s natural-language request and retrieved reference docs supply intent. For a fully-zeroed machine and a bring-up request the agent typically composes a multi-stage plan that walks beam through the chain; for a delivering-beam state and a trim-up request, a close-bounds plan with bounds shrunk around the current setpoint to a fraction of the safe range chosen by the agent from the live state, in practice converging near the $\sim 10\%$ bound established by earlier successful tunings rather than imposed by configuration; for an already-tuned machine that just needs phases reset, a small RF-phase-only plan against the booster-entry charge monitor.

Before submission, the agent validates the composed JSON plan against the optimizer’s input schema. Figure 2 shows the full loop: the operator approves the plan; `TuningScriptsMCP` dispatches the run; `tuning_scripts` executes the optimization within its own bounded envelope; and the structured run summary returns to the agent. The agent then either concludes or proposes a follow-up: a close-bounds refinement around the new working point, an al-

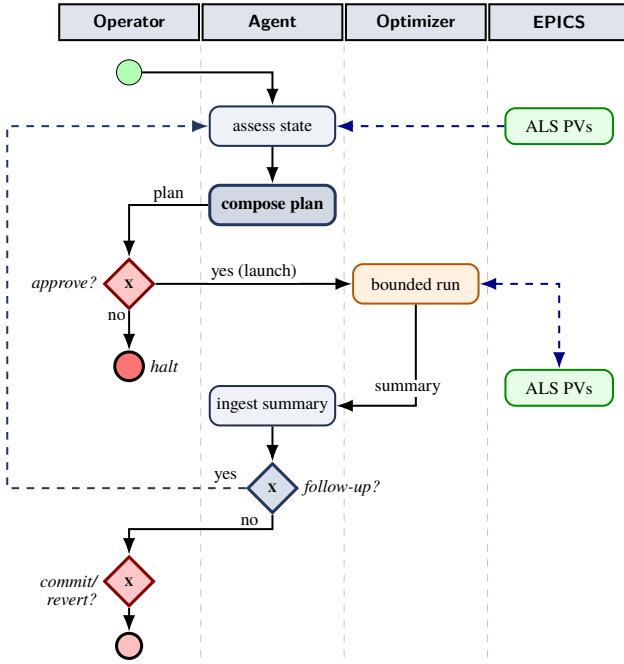


Figure 2: Orchestration loop across operator, agent, optimizer (tuning_scripts), and EPICS layers.

algorithm switch if the run failed to converge, or a bound widening. Each follow-up is itself a new plan-composition decision, gated again by operator approval, so the same logic chains across runs without per-run prompt engineering, and no agent-driven change becomes permanent without explicit operator commit at run termination.

RESULTS AT THE ALS

The results below come from an agent-driven session on the live ALS injector, dispatched from the Osprey runtime through TuningScriptsMCP. After assessing live state, the agent composed a close-bounds refinement plan: the 5 booster RF-phase knobs against the post-booster charge monitor (charge_bts), with bounds narrowed to $\pm 5\%$ of each variable's safe range around the current setpoint, and extremum seeking (ES) as the algorithm.

Figure 3 traces the launch payload and structured summary of the first run: at step 1 the agent reads charge readbacks and the 5 RF-phase readbacks through the EPICS MCP; at step 2 it retrieves the relevant reference documents and selects the close-bounds procedure consistent with the delivering-beam state; at step 3 it emits the schema-validated JSON configuration; the operator approves at step 4; the bounded ES run executes inside tuning_scripts at step 5; and the agent ingests the structured summary at step 6. Over 30 iterations, charge_bts rose from 0.629 nC at iteration 23 ($\sim 2.2\times$), and the machine ended the run at that setpoint.

On that summary the agent proposed a follow-up close-bounds refinement: same objective and algorithm, bounds re-centered on the new RF-phase setpoint. Routed through a fresh operator approval, the second run found no point better than its (already-optimal) starting one. The null result

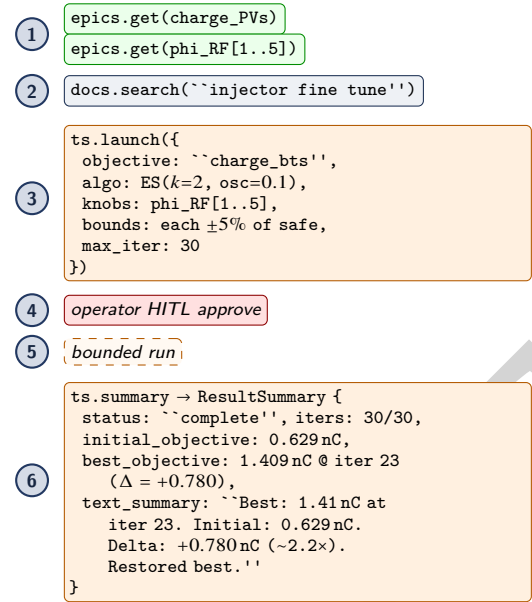


Figure 3: Trace of run 1 of an agent-driven close-bounds session on the live ALS injector, keyed to the steps of Fig. 2.

places a local maximum at charge_bts ≈ 1.3 nC at the current operating point. The agent ingested the summary and concluded the session without proposing another follow-up.

Operationally, this collapses operator involvement during a tuning session to high-level approval at the orchestration checkpoints (plan and configuration review, run dispatch, post-run accept/revert) rather than per-knob supervision of the optimizer's inner loop.

CONCLUSION AND OUTLOOK

We have shown that pairing the Osprey agentic framework with tuning_scripts yields autonomous orchestration of injector tuning at the ALS without weakening the safety envelope, by constraining the agent at the tool boundary rather than asking it to behave: every actuation passes through a mandatory HITL gate, every setpoint update inherits the bounded knob ranges tuning_scripts already enforces, and the agent itself never writes to EPICS. The same agent loop and generic optimizer port to any facility and control system by swapping the control-system I/O adapter and the capability scope, with no changes to the agent itself. Near-term work concentrates on a broader live-machine campaign, model-based prediction tools that let the agent anticipate run outcomes before committing, and tighter coupling with the drift-aware extensions of the underlying optimizer [5]. On the framework side, this is a prototype integration: once tuning_scripts is integrated into Osprey, EPICS access will flow through Osprey's control-system connector rather than a project-specific MCP.

ACKNOWLEDGEMENT

This work was supported by the Director of the Office of Science of the U.S. Department of Energy under Contract No. DEAC02-05CH11231.

REFERENCES

- [1] T. Hellert *et al.*, “Status of the Advanced Light Source”, in *Proc. IPAC'24*, Nashville, TN, USA, May 2024, pp. 1309–1312. [doi:10.18429/JACoW-IPAC2024-TUPG37](https://doi.org/10.18429/JACoW-IPAC2024-TUPG37)
- [2] A. Edelen and X. Huang, “Machine Learning for Design and Control of Particle Accelerators: A Look Backward and Forward”, *Annu. Rev. Nucl. Part. Sci.*, vol. 74, pp. 557–581, 2024. [doi:10.1146/annurev-nucl-121423-100719](https://doi.org/10.1146/annurev-nucl-121423-100719)
- [3] J. Kaiser, A. Lauscher, and A. Eichler, “Large language models for human-machine collaborative particle accelerator tuning through natural language”, *Sci. Adv.*, vol. 11, no. 1, p. eadr4173, 2025. [doi:10.1126/sciadv.adr4173](https://doi.org/10.1126/sciadv.adr4173)
- [4] A. Sulc, T. Hellert, R. Kammering, H. Hoschouer, and J. S. John, “Towards Agentic AI on Particle Accelerators”, 2025. [doi:10.48550/arXiv.2409.06336](https://doi.org/10.48550/arXiv.2409.06336)
- [5] G. Martino, T. Hellert, S. Leemann, and A. Sulc, “Hybrid On-line Optimization for Hands-off Tuning of the ALS Injector”, presented at IPAC'26, Deauville, France, May 2026, paper MOP6330, this conference.
- [6] T. Hellert, J. Montenegro, and A. Sulc, “Osprey: Production-ready agentic AI for safety-critical control systems”, *APL Mach. Learn.*, vol. 4, no. 1, p. 016103, Feb. 2026. [doi:10.1063/5.0306302](https://doi.org/10.1063/5.0306302)
- [7] T. Hellert *et al.*, “Status of Osprey: A Production Framework for Agentic AI in Control Systems”, presented at IPAC'26, Deauville, France, May 2026, paper MOP6322, this conference.
- [8] 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, no. 1, p. L012017, Jan. 2026. [doi:10.1103/jtqy-9jz1](https://doi.org/10.1103/jtqy-9jz1)
- [9] Model Context Protocol Specification, <https://modelcontextprotocol.io/specification>