SCPN Control: Compiling Stochastic Petri Nets into Spiking Neural Network Controllers for Real-Time Tokamak Plasma Control

Miroslav Sotek* and Michal Reiprich
ANULUM CH & LI

(Dated: February 27, 2026)

We present SCPN-CONTROL, an open-source neuro-symbolic control engine that compiles Stochastic Petri Nets (SPNs) into spiking neural network (SNN) controllers with formal contract verification for tokamak plasma control. The compilation pipeline translates SPN graphs into leaky integrate-and-fire (LIF) neuron pools with stochastic bitstream encoding, enforcing pre/post-condition contracts on every control observation and action. On a single CPU core the SNN controller achieves 11.9 µs median latency (84 kHz); with the Rust backend this drops to 2.1 µs (476 kHz), competitive with the DIII-D PCS physics loop rate of 4–10 kHz. The package integrates a 16-layer Kuramoto–Sakaguchi phase dynamics engine driven by the Paper 27 K_{nm} coupling matrix [1] and a Unified Phase Dynamics Equation (UPDE) multi-layer solver, monitored by a Lyapunov sliding-window stability guard. Five runtime-selectable controllers (PID, MPC, H_∞, SNN, neuro-cybernetic) operate on a shared Grad–Shafranov equilibrium solver with L-mode and H-mode profile support. Disruption prediction with shattered pellet injection (SPI) mitigation, a Gymnasium-compatible environment, and a digital twin complete the control stack. Benchmarked against published DIII-D and SPARC equilibria (GEQDSK format), the solver achieves $< 2\%$ RMSE on pressure and safety-factor profiles. The system (48 Python modules, 5 Rust crates, 701 tests, 17 CI jobs) is MIT/Apache-2.0 dual-licensed at `https://github.com/anulum/scpn-control`.

INTRODUCTION

Real-time plasma control in tokamaks demands sub-millisecond latency, formal safety guarantees, and runtime adaptability to regime transitions (L-mode to H-mode, ELM pacing, disruption avoidance). Existing plasma control systems (PCS) at DIII-D [2], ITER [3, 4], and EAST rely on classical PID or model-predictive control with hand-tuned gain schedules. These approaches lack formal verification of control actions and cannot adapt their control topology at runtime without operator intervention.

Recent work by Degrave et al. [5] demonstrated deep reinforcement learning for magnetic control on TCV, achieving multi-objective shaping with a single neural network policy. While impressive, the approach treats the controller as a black box—no formal contract checking is applied to outputs before they reach the plant, and the policy cannot be inspected for reachability or liveness properties.

Spiking neural networks (SNNs) offer event-driven, low-latency computation that maps naturally to real-time control tasks [6, 7]. Stochastic Petri Nets (SPNs) provide a formal framework for modeling concurrent, stochastic processes with well-defined reachability, boundedness, and liveness properties [8, 9].

We bridge these paradigms. SCPN-CONTROL compiles an SPN control specification into an SNN controller, preserving the formal contract structure of the Petri net as pre/post-condition checks on every control action. The SPN graph is declarative and inspectable; the compiled SNN is fast and neuromorphic-hardware-compatible.

Contributions.

1. A compilation pipeline from SPN graphs to LIF neuron pools with stochastic bitstream encoding, antithetic variance reduction, and formal contract verification (Section).

2. A 16-layer Kuramoto–Sakaguchi phase dynamics engine with the Paper 27 K_{nm} coupling matrix, UPDE multi-layer solver, PAC gating, and Lyapunov stability monitoring (Section).

3. Five runtime-selectable controllers (PID, MPC, H_∞, SNN, neuro-cybernetic) sharing a common contract interface, with a Gymnasium-compatible environment for RL benchmarking (Section).

4. A Rust-accelerated runtime achieving 2.1 µs P50 control latency (476 kHz), validated against DIII-D and SPARC equilibria at $< 2\%$ RMSE (Section).

5. Complete open-source release: 48 Python modules, 5 Rust crates, 701 tests, 17 CI jobs, Zenodo DOI (Section).

RELATED WORK

Classical PCS. The DIII-D PCS [2] runs real-time equilibrium reconstruction (EFIT) at 4–10 kHz on dedicated hardware, coupled with PID shape controllers. ITER's CODAC [3] specifies sub-millisecond plant I/O but relies on classical control algorithms. Walker and Humphreys [10] formalized vertical stability feedback but without runtime contract verification.

ML-based control. Degrave et al. [5] trained a deep RL policy for multi-objective TCV shape control. The policy achieves real-time execution but provides no formal guarantees on output bounds. TORAX [11] provides a differentiable JAX-based transport simulator suitable for gradient-based optimization but does not include a control stack.

Spiking neural networks. Third-generation neural networks [7] have been proposed for neuromorphic control but not previously applied to tokamak plasma control. Stochastic computing with bitstream encoding [12] enables low-precision, fault-tolerant arithmetic suited to noisy plasma environments.

Petri nets in control. Petri nets have been used for supervisory control and discrete-event systems [9] but not previously compiled into neural network controllers. Our SPN-to-SNN compilation is, to our knowledge, the first such pipeline applied to continuous plasma control.

SPN TO SNN COMPILATION

Stochastic Petri Net Specification

A Stochastic Petri Net is a tuple $\mathcal{N} = (P, T, A, W, M_0, \Lambda)$ where P is a finite set of places, T a finite set of transitions, $A \subseteq (P \times T) \cup (T \times P)$ the arc set, $W : A \to \mathbb{N}$ the arc weights, $M_0 : P \to \mathbb{N}_0$ the initial marking, and $\Lambda : T \to \mathbb{R}_{>0}$ the stochastic firing rates.

Places represent physical observables (I_p, W_{MHD}, $\bar{n}_e$, vertical position Z) and controller states. Transitions encode control decisions (heating power, fueling, vertical stability corrections). The SPN is specified declaratively; the user does not write neural network code.

Compilation to LIF Neurons

The `FusionCompiler` maps each transition $t_j \in T$ to a stochastic leaky integrate-and-fire (LIF) neuron. The marking $M(p_i)$ of each place is encoded as a Bernoulli bitstream of length L:

$$b_i^{(k)} \sim \text{Bernoulli}\left(\frac{M(p_i)}{M_{\max}}\right), \quad k = 1, \ldots, L \tag{1}$$

where $L \geq 64$ and $M_{\max}$ normalizes markings to $[0, 1]$. The bitstreams are packed into `uint64` words ($n_{\text{words}} = \lceil L/64 \rceil$) with per-element splitmix64 seed derivation for statistical independence.

Arc weights $W(p_i, t_j)$ define two incidence matrices: $\mathbf{W}_{\text{in}} \in \mathbb{R}^{|T|\times|P|}$ (consumption) and $\mathbf{W}_{\text{out}} \in \mathbb{R}^{|P|\times|T|}$ (production). Each LIF neuron integrates:

$$V_j(t+1) = V_j(t) + \frac{\Delta t}{\tau_m}\big(V_{\text{rest}} - V_j(t)\big) + I_j(t) + \sigma\,\eta_j(t) \tag{2}$$

where $I_j = [\mathbf{W}_{\text{in}}\,\mathbf{m}]_j$ is the synaptic input from the current marking $\mathbf{m}$, $\tau_m = 10$ ms, and $\sigma = 0.1$ is the noise amplitude. When $V_j > V_{\text{th}}$, the neuron fires ($f_j = 1$) and resets to $V_{\text{reset}} = 0$.

The marking update rule preserves the Petri net semantics:

$$\mathbf{m}_{k+1} = \text{clip}\big(\mathbf{m}_k - \mathbf{W}_{\text{in}}^{\top}\mathbf{f}_k + \mathbf{W}_{\text{out}}\,\mathbf{f}_k,\ 0,\ 1\big) \tag{3}$$

Algorithm summarizes the compilation and Algorithm the runtime tick.

Algorithm 1: SPN → SNN Compilation

```
Require: SPN $\mathcal{N} = (P, T, A, W, M_0, \Lambda)$, bitstream length $L$
1: Build $\mathbf{W}_{\text{in}}$, $\mathbf{W}_{\text{out}}$ from arc set $A$ and weights $W$
2: for each transition $t_j \in T$ do
3:     Create LIF neuron with $V_{\text{th}} = \Lambda(t_j)$
4: end for
5: Pack $\mathbf{W}_{\text{in}}$ into uint64 bitstreams (Eq. 1)
6: Attach contracts from ControlObservation, ControlAction
7: return CompiledNet ($\mathbf{W}_{\text{in}}, \mathbf{W}_{\text{out}}$, neurons, contracts)
```

Algorithm 2: Control Tick (Runtime)

```
Require: CompiledNet, observation $\mathbf{o}_t$, targets $\mathbf{r}$, marking $\mathbf{m}_t$
1: Pre-check: verify $\mathbf{o}_t$ within physical bounds
2: $\mathbf{e}_t \leftarrow (\mathbf{r} - \mathbf{o}_t)/\mathbf{s}$                ▷ normalized error
3: Inject $\mathbf{e}_t$ into designated places
4: $\mathbf{I} \leftarrow \mathbf{W}_{\text{in}}\,\mathbf{m}_t$                      ▷ synaptic input
5: $\mathbf{f}_t \leftarrow$ LIF_step($\mathbf{I}$)                  ▷ fire vector, Eq. 2
6: $\mathbf{m}_{t+1} \leftarrow$ Eq. 3
7: $\mathbf{a}_t \leftarrow$ decode($\mathbf{m}_{t+1}$)          ▷ push-pull + slew + clamp
8: Post-check: verify $\mathbf{a}_t$ within actuator limits
9: return $\mathbf{a}_t$, $\mathbf{m}_{t+1}$
```

Stochastic Computing Acceleration

The dense forward pass $\mathbf{y} = \mathbf{W}\,\mathbf{x}$ can be computed via stochastic bitstream arithmetic: input bitstreams are AND-ed with weight bitstreams, and the output is the popcount:

$$y_i = \frac{1}{L}\sum_{k=1}^{L} \text{popcount}\Big(\mathbf{w}_i^{(k)} \,\&\, \mathbf{x}^{(k)}\Big) \tag{4}$$

With `sc-neurocore` $\geq$ 3.8.0, this uses `VectorizedSCLayer` with a Rust SIMD backend, achieving 512× real-time throughput via packed `uint64` operations.

Antithetic variance reduction. For n stochastic passes, we use antithetic sampling: for each base sample $u \sim \text{Uniform}(0, 1)$, we evaluate both $u < p$ and $u > 1-p$, halving the variance of the firing probability estimator at no additional memory cost.

Formal Contract Verification

Every control tick verifies two contract layers:

Observation contracts. The observation $\mathbf{o}_t = (R_{\text{axis}}, Z_{\text{axis}})$ is checked against physical bounds. Errors are normalized and clipped to $[-1, 1]$ before injection into the SPN marking.

Action contracts. The decoded action $\mathbf{a}_t = (\Delta I_{\text{PF3}}, \Delta I_{\text{PF,tb}})$ is subject to slew-rate limiting ($|\dot{u}| \leq \dot{u}_{\max}$) and absolute saturation ($|u| \leq u_{\text{abs}}$). Violations trigger a safe fallback (zero-action or last-known-good).

Physics invariants. Five runtime invariants are monitored: $q_{\min} > 1$ (Kruskal–Shafranov [13]), $\beta_N < 2.8$ (Troyon limit [14]), Greenwald fraction < 1.2 [15], $T_i < 25$ keV (first-wall constraint), and $|\Delta W/W| < 0.01$ (energy conservation). Violations exceeding 20% of the threshold are classified as critical and trigger immediate protective action.

PHASE DYNAMICS ENGINE

Kuramoto–Sakaguchi with Global Field Driver

Following the Kuramoto model [16] with the Sakaguchi phase frustration extension [17] and SCPN Paper 27 [1], we model N coupled oscillators across $L = 16$ layers:

$$\dot{\theta}_n^{(\ell)} = \omega_n + \frac{1}{N_\ell} \sum_{m=1}^{N_\ell} K_{\ell\ell} \sin\Big(\theta_m^{(\ell)} - \theta_n^{(\ell)} - \alpha\Big) + \zeta_\ell \sin\Big(\Psi - \theta_n^{(\ell)}\Big) \tag{5}$$

where ω_n are natural frequencies drawn from the Paper 27 canonical set, $K_{\ell\ell'}$ is the inter-layer coupling matrix, α is the Sakaguchi phase-lag parameter, ζ_ℓ is the per-layer global field coupling strength, and Ψ is the mean-field phase [18].

The order parameter measures synchronization:

$$R_\ell \, e^{i\Psi_\ell} = \frac{1}{N_\ell} \sum_{n=1}^{N_\ell} e^{i\theta_n^{(\ell)}} \tag{6}$$

$R_\ell = 1$ indicates full synchronization; $R_\ell \to 0$ indicates incoherence.

Paper 27 K_{nm} Coupling Matrix

The 16×16 coupling matrix encodes inter-layer interactions with exponential distance decay and empirical calibration anchors:

$$K_{nm} = K_{\text{base}} \, e^{-\alpha_K |n-m|} + \sum_{(i,j)\in\mathcal{A}} c_{ij} \, \delta_{ni} \delta_{mj} \tag{7}$$

with $K_{\text{base}} = 0.45$, decay $\alpha_K = 0.3$, and calibration anchors $\mathcal{A}$:

(n,m)	$(1,2)$	$(2,3)$	$(3,4)$
c_{nm}	0.302	0.201	0.252
(n,m)	$(4,5)$	$(1,16)$	$(5,7)$
c_{nm}	0.154	≥ 0.05	≥ 0.15

The last two entries are cross-hierarchy boosts (L1↔L16 and L5↔L7) from Paper 27 §4.3.

UPDE Multi-Layer Solver

The Unified Phase Dynamics Equation (UPDE) couples all 16 layers. For oscillator n in layer ℓ:

$$\dot{\theta}_n^{(\ell)} = \omega_n^{(\ell)} + \frac{1}{N_\ell} \sum_m K_{\ell\ell} \sin\Big(\theta_m^{(\ell)} - \theta_n^{(\ell)}\Big) + \sum_{\ell' \neq \ell} K_{\ell\ell'} \, R_{\ell'} \sin\Big(\Psi_{\ell'} - \theta_n^{(\ell)}\Big) + \zeta_\ell \sin\Big(\Psi - \theta_n^{(\ell)}\Big) \tag{8}$$

PAC gating. Optional phase-amplitude coupling modulates cross-layer drive based on source coherence:

$$\tilde{K}_{\ell\ell'} = K_{\ell\ell'} \left[1 + \gamma_{\text{PAC}} \left(1 - R_{\ell'}\right)\right] \tag{9}$$

When source layer ℓ' desynchronizes ($R_{\ell'} \downarrow$), inter-layer coupling increases, implementing a stabilizing negative feedback. The overhead is ~12% (~98 µs per step on 16×256 oscillators).

Lyapunov Stability Guard

A sliding-window monitor tracks the Lyapunov candidate function [18]:

$$V(t) = \frac{1}{N} \sum_{n=1}^{N} \left(1 - \cos(\theta_n - \Psi)\right) \tag{10}$$

and computes the exponential rate over a configurable window of $W = 50$ samples:

$$\lambda = \frac{1}{\Delta t} \ln\left(\frac{V(t)}{V(t - W\Delta t)}\right) \tag{11}$$

$\lambda < 0$ indicates convergence toward synchronization. If $\lambda > 0$ for $K_{\max} = 3$ consecutive windows, the guard triggers a coherence recovery action (increased ζ or controller gain boost).

Table I shows λ as a function of global field strength ζ for $N = 1000$ oscillators.

CONTROL ARCHITECTURE

Controller Suite

All five controllers share the contract interface of Section : observation pre-check, action post-check, and physics invariant monitoring.

TABLE I. Lyapunov exponent λ vs. global field strength ζ ($N = 1000$, 200 steps, $\Delta t = 1$ ms).

ζ	λ ($K = 0$)	λ ($K = 2$)
0.0	+0.01	+0.04
0.1	−0.03	−0.02
0.5	−0.23	−0.24
1.0	−0.49	−0.53
3.0	−1.65	−1.83
5.0	−3.01	−3.35

PID. Dual-channel I_p + vertical position control with anti-windup and derivative low-pass filtering.

MPC. Gradient-descent optimization over a 10-step prediction horizon with L_2 regularization ($\lambda = 0.1$):

$$J = \sum_{t=0}^{N_p-1} \|\mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t - \mathbf{x}_{\text{ref}}\|^2 + \lambda \|\mathbf{u}_t\|^2 \tag{12}$$

The plant model is a linearized 4-state system ($\mathbf{x} = [R_{\text{axis}}, Z_{\text{axis}}, R_x, Z_x]^\top$) with coil-current inputs. Action limits are ±2.0 A per coil per step.

H_∞. γ-iteration robust synthesis [19, 20] on a linearized vertical stability plant:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}_1\mathbf{w} + \mathbf{B}_2\mathbf{u} \tag{13}$$

$$\mathbf{z} = \mathbf{C}_1\mathbf{x} + \mathbf{D}_{12}\mathbf{u} \tag{14}$$

The controller minimizes $\|\mathbf{T}_{zw}\|_\infty < \gamma$ via bisection on γ (range $[1.01, 10^6]$, $r_{\text{tol}} = 10^{-3}$, max 100 iterations). State-feedback and observer gains are obtained from the continuous algebraic Riccati equations; discrete-time adaptation uses zero-order-hold discretization. Robustness margin: ±20% multiplicative plant uncertainty.

SNN. Compiled from SPN specification (Section).

Neuro-cybernetic. Dual radial (R) + vertical (Z) spiking neuron pools ($n = 20$ neurons each) with push-pull decoding:

$$u = g \cdot \left(\frac{N_+}{N_w} - \frac{N_-}{N_w}\right) \tag{15}$$

where $N_\pm$ are spike counts in the positive/negative populations over a window of N_w ticks and g is the output gain.

Grad–Shafranov Equilibrium Solver

The GS equation in toroidal coordinates:

$$\Delta^*\psi \equiv R\frac{\partial}{\partial R}\left(\frac{1}{R}\frac{\partial\psi}{\partial R}\right) + \frac{\partial^2\psi}{\partial Z^2} = -\mu_0 R^2 \frac{dp}{d\psi} - \frac{1}{2}\frac{dF^2}{d\psi} \tag{16}$$

is solved by Picard iteration on an $N_R \times N_Z$ grid (default 65×65).

Vacuum field. For each coil at (R_c, Z_c) with current I_c:

$$\psi_{\text{vac}} = \frac{\mu_0 I_c}{2\pi}\sqrt{(R + R_c)^2 + \Delta Z^2}\,\frac{(2 - k^2)K(k) - 2E(k)}{k^2} \tag{17}$$

where $k^2 = 4RR_c/[(R+R_c)^2 + \Delta Z^2]$ and K, E are complete elliptic integrals.

Profile modes. L-mode: $p(\rho) = p_0(1 - \rho^{\alpha_c})$. H-mode: core profile with mtanh pedestal transition at ρ_{ped} with configurable height and width.

Convergence to $\|\delta\psi\|_\infty < 10^{-6}$ typically in 12–18 iterations. Solver backends: Python (SOR, Picard), Rust (multigrid + SOR), or optional C++ HPC bridge.

Disruption Prediction and SPI Mitigation

An 11-dimensional feature vector is extracted from the plasma state at each tick: mean, std, max, and slope of the tearing mode signal, energy ($\langle s^2 \rangle$), last value, toroidal mode amplitudes ($n = 1, 2, 3$), toroidal asymmetry index, and radial spread.

The tearing mode physics follows the modified Rutherford equation [21]:

$$\frac{dw}{dt} = \Delta'(1 - w/w_{\text{sat}}) \tag{18}$$

Disruption is triggered when the island width exceeds $w > 8.0$ (mode lock).

An LSTM classifier (PyTorch, optional) predicts P(disrupt). When $P > 0.8$, shattered pellet injection (SPI) [22] is triggered with halo current and runaway electron post-disruption physics.

The predictor is trained on synthetic data derived from 11 reference DIII-D shot profiles [23] (6 anomalous: locked mode, density limit, VDE, tearing, beta limit; 5 safe: hybrid, H-mode, snowflake, negative delta, high beta).

Digital Twin and Gymnasium Environment

TokamakDigitalTwin. A 2D diffusive-reaction model on a 40 × 40 poloidal grid with core Gaussian heating (+5.0 K/step), configurable turbulent diffusivity ($D_{\text{turb}} = 0.5$ in island regions), and radiative losses ($\propto T^2$). Rational surface detection at $q = 1.5, 2.0, 2.5, 3.0$ marks regions of enhanced transport.

TokamakFlightSim. Isoflux flight simulator with first-order actuator dynamics ($\tau = 60$ ms), rate limits ($\leq 10^6$ A/s, ITER PF coil spec), sensor noise, and configurable measurement delay.

TokamakEnv. Gymnasium-compatible [24] environment wrapping the digital twin. Observation space:

TABLE II. Control loop latency (5000 iterations, single core).

Controller	P50 (µs)	P99 (µs)	Throughput
PID	3.2	8.1	312 kHz
SNN (NumPy)	11.9	28.4	84 kHz
SNN (Rust)	2.1	5.3	476 kHz
MPC (N_p=10)	142	310	7.0 kHz
H_∞	89	195	11.2 kHz

TABLE III. Comparison with open-source fusion software.

Code	GS	RT Control	Language
DIII-D PCS [2]	Yes	PID	C/Fortran
TORAX [11]	Yes	—	JAX
FreeGS [25]	Yes	—	Python
DeepMind [5]	—	RL	TF/JAX
SCPN-CONTROL	Yes	5 types	Py + Rust

4D continuous (R, Z, R_x, Z_x). Action space: continuous coil currents. Reward: $-\|\mathbf{x} - \mathbf{x}_{\mathrm{ref}}\|^2$ with disruption penalty.

BENCHMARKS AND VALIDATION

Control Loop Timing

Table II reports median (P50) and tail (P99) latencies measured over 5000 iterations on a single AMD Ryzen core.

The Rust SNN backend (476 kHz) exceeds the DIII-D PCS physics loop rate of 4–10 kHz by ∼50×. The MPC and H_∞ controllers, while slower, still meet real-time requirements for the 1–10 kHz control bands typical in present-day experiments.

Competitive Comparison

Table III compares SCPN-CONTROL against open-source fusion codes.

SCPN-CONTROL is the only open-source package combining equilibrium solving, multiple real-time controller types (including SNN), formal contract verification, disruption prediction, and a Gymnasium environment in a single installation.

Kuramoto Phase Sync: Python vs. Rust

Table IV reports Kuramoto step latency for increasing oscillator count N.

At N = 16,384 the Rust backend achieves sub-millisecond Kuramoto steps (0.544 ms), enabling real-time UPDE solving for large oscillator populations.

TABLE IV. Kuramoto step latency: Python vs. Rust.

N	Python (ms)	Rust (ms)	Speedup
64	0.050	0.003	17×
1,000	0.087	0.062	1.4×
4,096	0.328	0.180	1.8×
16,384	1.240	0.544	2.3×
65,536	5.010	1.980	2.5×

TABLE V. Normalized RMSE against reference equilibria.

Profile	DIII-D	SPARC
$\psi(R, Z)$	$< 1\%$	$< 1.5\%$
$p(\rho)$	$< 2\%$	$< 2\%$
$q(\rho)$	$< 2\%$	$< 2.5\%$

Equilibrium RMSE

Table V reports RMSE against published DIII-D shot 164984 and SPARC v2c equilibria in GEQDSK format.

These RMSE values are CI-gated: if any regression exceeds the threshold, the CI pipeline fails and the commit is rejected.

Phase Dynamics Convergence

With $N = 50$ oscillators per layer, $L = 16$ layers, $\zeta = 0.5$, and the Paper 27 K_{nm} matrix: the order parameter R converges from $R(0) \approx 0.15$ to $R(500) = 0.92$ within 500 ticks. The Lyapunov exponent λ (Eq. 11) settles to -0.47, confirming asymptotic stability of the synchronized state.

SOFTWARE ARCHITECTURE

Python Package

48 modules organized into four subpackages: `scpn/` (5 modules: SPN graph, compiler, contracts, controller, safety interlocks), `core/` (11 modules: GS solver, transport, scaling laws, EQDSK I/O, uncertainty), `control/` (17 modules: 5 controller types, digital twin, flight sim, Gymnasium env, disruption predictor), `phase/` (7 modules: Kuramoto, UPDE, K_{nm}, Lyapunov guard, realtime monitor, WebSocket server). Core dependencies: `numpy` $\geq$ 1.24, `scipy` $\geq$ 1.10, `click` $\geq$ 8.0. All other dependencies (torch, matplotlib, streamlit, h5py, websockets, sc-neurocore, nengo) are optional extras.

Rust Workspace

Five crates compiled via PyO3 [26]:

1. **control-types**: `PlasmaState`, `EquilibriumConfig`, `ControlAction`, physical constants.
2. **control-math** (14 modules): multigrid, SOR, GMRES, Chebyshev, FFT, Boris pusher, LIF neuron, Kuramoto step, symplectic integrator, tridiagonal solver.
3. **control-core** (20 modules): GS kernel, transport, vacuum field, X-point detection, inverse solver, pedestal, stability, RF heating, VMEC/BOUT++ interfaces.
4. **control-control** (10 modules): PID, MPC, H_∞, SNN pool, SPI mitigation, digital twin, optimal control, SOC learning.
5. **control-python**: PyO3 bindings exporting `PyFusionKernel`, `PySnnPool`, `PyMpcController`, `PyPlasma2D`, `PyTransportSolver`, `PyRealtimeMonitor`, and SCPN kernels (`dense_activations`, `marking_update`, `sample_firing`).

Python code auto-detects the Rust backend at import time via `importlib.util.find_spec` and dispatches hot-path calls accordingly, with a pure-Python fallback for all functionality.

Testing and CI

701 tests across 50 files, 61% line coverage, enforced by 17 CI jobs:

- **Python**: pytest, ruff, mypy, bandit, coverage ($\geq$ 55% gate), RMSE regression gates (DIII-D, SPARC, ITER)
- **Rust**: `cargo test`, clippy, fmt, audit, Criterion benchmarks (Kuramoto, transport, LIF, Boris)
- **Integration**: phase-sync benchmark ($N = 1000, 4096$), docs build (MkDocs)

LIMITATIONS

We state limitations explicitly so they are not overstated by inference:

1. **Equilibrium**: Fixed-boundary Grad–Shafranov only. Free-boundary with external coil currents is not implemented. No stellarator geometry.
2. **Transport**: 1.5D flux-surface-averaged with Chang–Hinton neoclassical + scaling-law anomalous transport. No TGLF or QuaLiKiz microinstability models.
3. **Validation**: Benchmarked against analytic Solov'ev solutions and published DIII-D/SPARC GEQDSK files. No real MDSplus shot data from active experiments.
4. **Disruption predictor**: Trained on synthetic data only. Not validated on experimental disruption databases (e.g., DIII-D disruption DB, JET disruption DB).
5. **Deployment**: Research-grade software. Not hardened for production PCS integration. No ITER CODAC, EPICS, or FPGA interface.
6. **Rust acceleration**: Optional. Pure-Python fallback is 5–10× slower for GS solve and 2–3× slower for Kuramoto steps at $N > 1000$.

CONCLUSION AND FUTURE WORK

We presented a neuro-symbolic approach to tokamak plasma control that compiles Stochastic Petri Nets into spiking neural network controllers through a formal pipeline preserving contract verification at every control tick.

The key technical results are: (i) control logic is specified declaratively as a Petri net graph and compiled automatically into LIF neuron pools; (ii) every control action is contract-verified before reaching the plant, with five physics invariants monitored at runtime; (iii) the Rust-accelerated SNN achieves 2.1 µs P50 latency, exceeding the DIII-D PCS rate by $\sim$50×; (iv) the 16-layer UPDE phase engine with Lyapunov monitoring converges to $R = 0.92$ (from $R_0 = 0.15$) with $\lambda = -0.47$ under Paper 27 K_{nm} coupling.

Future work targets:

- Free-boundary GS with external coil current optimization
- TGLF/QuaLiKiz turbulence closure via neural surrogates
- Validation on real MDSplus shot data (DIII-D, EAST)
- FPGA deployment via the `sc-neurocore` HDL backend for sub-microsecond latency
- Integration with IMAS/OMAS for cross-device portability

The software is MIT/Apache-2.0 dual-licensed and available at `https://github.com/anulum/scpn-control` with a live dashboard at `https://scpn-control.streamlit.app` and archived at `https://doi.org/10.5281/zenodo.18804940`.

This work builds on the SCPN theoretical framework (Papers 1–27) and the `scpn-fusion-core` codebase. We thank the open-source fusion community for reference equilibria, the DIII-D team for published GEQDSK data, and the Commonwealth Fusion Systems team for SPARC equilibrium parameters.

* protoscience@anulum.li

[1] M. Sotek, (2026), sCPN Paper 27, `https://www.academia.edu/143833534/27_SCPN_The_Knm_Matrix`.

[2] J. R. Ferron, M. L. Walker, L. L. Lao, H. E. St John, D. A. Humphreys, and J. A. Leuer, Nucl. Fusion **38**, 1055 (1998).

[3] ITER Organization, *CODAC Core System Overview*, Tech. Rep. (ITER Organization, 2020) iTER_D_2V3V8G.

[4] D. A. Humphreys, T. A. Casper, N. Eidietis, M. Ferrara, D. A. Gates, I. H. Hutchinson, G. L. Jackson, E. Kolemen, J. A. Leuer, J. Lister, *et al.*, Phys. Plasmas **22**, 021806 (2015).

[5] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, *et al.*, Nature **602**, 414 (2022).

[6] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, Neural Networks **111**, 47 (2019).

[7] W. Maass, Neural Networks **10**, 1659 (1997).

[8] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets* (Wiley, 1995).

[9] T. Murata, Proc. IEEE **77**, 541 (1989).

[10] M. L. Walker and D. A. Humphreys, Automatica **45**, 665 (2009).

[11] A. Fanni, J. Citrin, F. Felici, *et al.*, Nucl. Fusion (2024), arXiv:2406.06718.

[12] A. Alaghi and J. P. Hayes, ACM Trans. Embed. Comput. Syst. **12**, 1 (2013).

[13] J. P. Freidberg, *Ideal MHD* (Cambridge University Press, 2014).

[14] F. Troyon, R. Gruber, H. Saurenmann, S. Semenzato, and S. Succi, Plasma Phys. Control. Fusion **26**, 209 (1984).

[15] M. Greenwald, Plasma Phys. Control. Fusion **44**, R27 (2002).

[16] Y. Kuramoto, Lecture Notes in Physics **39**, 420 (1975).

[17] H. Sakaguchi and Y. Kuramoto, Prog. Theor. Phys. **76**, 576 (1986).

[18] J. A. Acebrón, L. L. Bonilla, C. J. Pérez Vicente, F. Ritort, and R. Spigler, Rev. Mod. Phys. **77**, 137 (2005).

[19] K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control* (Prentice Hall, 1996).

[20] K. Glover and J. C. Doyle, Syst. Control Lett. **11**, 167 (1988).

[21] P. C. de Vries, M. F. Johnson, B. Alper, P. Buratti, T. C. Hender, H. R. Koslowski, and V. Riccardo, Nucl. Fusion **51**, 053018 (2011).

[22] M. Lehnen, K. Aleynikova, P. B. Aleynikov, D. J. Campbell, P. Drewelow, N. W. Eidietis, Y. Gasber, R. S. Granetz, Y. Gribov, N. Hartmann, *et al.*, J. Nucl. Mater. **463**, 39 (2015).

[23] C. Rea and R. S. Granetz, Fusion Sci. Technol. **76**, 912 (2019).

[24] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. de Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, *et al.*, Gymnasium (2023), `https://github.com/Farama-Foundation/Gymnasium`.

[25] B. Dudson *et al.*, FreeGS: Free-boundary Grad–Shafranov solver (2023), `https://github.com/freegs-plasma/freegs`.

[26] PyO3 Contributors, PyO3: Rust bindings for Python (2024), `https://pyo3.rs`.