# Autotuning and Self-Referential Dynamics in a Minimal Bistable Workspace

Brandon Raeder et al.

November 9, 2025

**Abstract**

We present a compact software environment ("lab.py") for experimenting with three recurrent dynamical architectures: (A) a bistable network modulated by a global workspace signal, (B) a reflective hierarchical "why-loop" architecture, and (C) a self-referential workspace that predicts and modulates its own compressed state representation. A lightweight meta-autotuning process proposes parameter updates (gain $\alpha$ and workspace coupling $\epsilon$) using a small neural meta-tuner and rollout-based causal reward. To maintain rich dynamics and prevent resonance lock-in, we introduce a dual-frequency perturbation with slow drift. The system includes instrumentation for entropy, a Lyapunov proxy, a Lempel–Ziv complexity proxy, and mutual information, along with an interactive GUI for visual inspection. This document summarizes implementation details and early smoke-test results.

## 1 Introduction

Understanding how global coupling and self-modeling interact with bistable local dynamics is relevant for exploring mechanisms of coherent processing, surprise, and meta-level regulation in minimal neural systems. To this end we developed `lab.py`, a lightweight and reproducible environment supporting rapid iteration, interactive visualization, automated smoke tests, and parameter autotuning guided by complexity-based reward.

## 2 Methods

### 2.1 Model Variants

We implemented three configurations:

- **Option A**: $N$ bistable units with a global workspace variable $ws$ that integrates mean population activity and feeds back with strength $\epsilon$.
- **Option B**: Two coupled bistable cascades with an additional reflective "why" unit driving workspace activity through a saturating map.
- **Option C**: A self-referential workspace maintaining a compressed self-model vector (mean, variance, trend, normalized workspace, short-term entropy, coherence) along with a small predictor that forecasts future self-model values. Prediction error modulates local dynamics.

All models use $\tanh(\alpha x - \theta)$ as the bistable activation and Euler integration with timestep $dt$.

### 2.2 Autotuning

A small PyTorch network maps complexity features $[\text{entropy}, R, \text{lyap}, \text{complexity}]$ to normalized parameters $[\hat{\alpha}, \hat{\epsilon}]$. Background workers:

1. Propose candidate parameters and run short rollouts.
2. Compute reward: $\text{reward} = w_H \cdot H + w_R \cdot R_{\text{final}}$.
3. Store $(\text{features}, \text{params}, \text{reward})$ in an experience buffer and update the meta-tuner by reward-weighted regression.

A fallback heuristic based on entropy is used if PyTorch is unavailable.

### 2.3 Irrational-Time Perturbation

To prevent resonance lock-in, low-amplitude sinusoidal perturbations at two incommensurate periods $(\tau_1, \tau_2)$ are introduced with a slow phase drift.

## 3 Implementation Notes

The codebase is contained in a single script (`lab.py`) plus a helper file for self-model utilities.

- **GUI**: A 3×2 layout provides workspace heatmaps and phase traces with rolling-window views.
- **Threading**: Autotuning threads are started only after Matplotlib animations are active to avoid Tk race conditions.
- **Metrics**: Shannon entropy, Lyapunov-step proxy, LZ-76 complexity on thresholded traces, pairwise mutual information, and simple linear decoders.
- **Testing**: `smoke_test_autotune()` validates background worker updates and buffer behavior.

## 4 Results

Preliminary runs indicate:

- The experience buffer is actively populated and updates the meta-tuner toward higher-entropy regimes.
- The dual-frequency perturbation reduces overly persistent oscillatory modes relative to single-frequency perturbation.
- Deferring autotune thread initialization until after GUI setup prevents Tk/Tcl instability.

## 5 Discussion

Short rollouts combined with a reward-weighted experience buffer provide an effective causal training signal for parameter tuning. Option C introduces a self-modeling signal via prediction error, adding a channel for dynamic self-regulation. Metrics remain proxy-based and require further validation across longer runs and broader parameter sweeps.

Table 1: Example metrics from a short 500-step run (illustrative only).

| Model | Mean $R$ | Entropy | Lyap Proxy | Reward |
|---|---|---|---|---|
| Option A | 0.12 | 1.97 | 0.032 | 0.45 |
| Option B | 0.08 | 2.14 | 0.041 | 0.62 |
| Option C | 0.20 | 2.35 | 0.029 | 0.71 |

## 6 Reproducibility

Dependencies: Python 3.10+, NumPy, Matplotlib, scikit-learn, PyTorch (optional).

```
# GUI (interactive)
python3 lab.py

# Headless smoke test
python3 -c "import matplotlib; matplotlib.use('Agg'); \
             import lab; lab.smoke_test_autotune(2.0)"
```

## 7 Conclusion

This environment offers a compact platform for exploring workspace-mediated dynamics, self-modeling, and automatic parameter adjustment. Its minimal footprint is intended to encourage experimentation and modification.

## Acknowledgements

Development and testing were performed by the project team. Code is available at: `https://github.com/Brandonl`

## References

[1] G. Tononi, *Consciousness as Integrated Information*, Biological Bulletin, 2008.

[2] D. Wolpert, Z. Ghahramani, M. Jordan, *An internal model for sensorimotor integration*, 1995.