DOCUMENTATION

The SNN tool

ar011 · 6 July 2026

One command-line tool, *tools/snn/tool.py*, drives every simulation, training run, and measurement in this project. Experiment runners never import the model code; they shell out to this tool, which writes data files (metrics, rasters, population traces, weight dumps) into an output directory, and then draw their own figures from those files. The tool is the engine; the plotting lives in the runners under *experiments/*.

This page is the reference for that engine. It is not a tutorial. It is the dictionary you reach for when an experiment mentions *–ei-strength 0.5* and you want to know exactly what that did.

At a glance

- Run it with *uv run python tools/snn/tool.py* (never a bare *python*; the repo convention is *uv*).
- Three subcommands: *sim* (forward pass plus metrics), *train* (surrogate-gradient BPTT), *dump-weights* (emit weight matrices).
- Every parameter is a flag. There is no global config file; a saved run's *config.json* can be inherited with *–load-config*.
- Every run writes *config.json*, *run.sh*, *output.log*, and *run.jsonl* for reproducibility.
- The tool emits data, not figures. The runner draws the figure.

The tool and the experiment

demolab draws a hard line between a **tool** and an **experiment**. A tool holds the reusable science and speaks only through files; an experiment (a runner in *experiments/expNNN.py*) chooses which tool commands to run, then reads their data files back and renders the figures. The runner reaches the tool by running its CLI as a subprocess, never by importing it. That firewall is what keeps *tools/snn/* generic and lets the same engine serve every writeup in the lab.

So a typical experiment does three things: it invokes *tool.py* (often many times, sweeping a parameter), it aggregates each run's config plus headline metrics into a single *numbers.json*, and it draws PNG or SVG figures from the run's data. The committed record lands in *artifacts/data/expNNN/*; the tool's own scratch output is disposable (see Artifacts).

Quick start

Run a metrics-only forward pass of the canonical PING network:

```
uv run python tools/snn/tool.py sim --model ping --ei-strength 0.5
```

Train on MNIST for 100 epochs:

```
uv run python tools/snn/tool.py train --dataset mnist --epochs 100 --lr
0.0001
```

Evaluate a trained run on the test set, replaying it at a coarser timestep:

```
uv run python tools/snn/tool.py sim --infer \
  --load-config runs/foo/config.json \
  --load-weights runs/foo/weights.pth \
  --dt 0.5
```

Each subcommand has its own complete *–help*, and that per-subcommand help is authoritative. The top-level dispatcher's group summary is hand-maintained and can lag the parser. When a flag here disagrees with reality, run *sim –help* and trust that.

Commands

sim

Run one forward pass and report firing-rate metrics. This is the cheapest mode (no training, no plots), used by the test suite, the dt-stability checks in ar003, and every experiment that needs a single inference pass over a trained or fresh network.

```
uv run python tools/snn/tool.py sim --model ping --dataset mnist --digit 3
```

On its own it prints metrics. The flags below make it load trained weights, evaluate a test set, emit extra data artifacts, or inject perturbations. There is no *–image* or *–video*: where those retired flags once produced panels and sweep MP4s, *sim* now emits raw data (via *–outputs*) that the calling runner plots, and sweep videos are assembled runner-side from many *sim* calls.

flag	default	description
–infer	off	Load trained weights and evaluate test-set accuracy; writes *results.json* (and *metrics.json*).
–load-config PATH	—	Load a saved *config.json* and inherit its model, dataset, and parameters. Explicit CLI flags override loaded values.
–load-weights PATH	—	Path to a *weights.pth* file for inference.

–max-samples INT	all	[*–infer*] Cap the evaluation set to N samples.
–outputs OUTPUT [dots.c]	—	[*–infer*] Extra artifacts from the single forward pass (*metrics.json* always written): *per_cell_rates* (per-cell E/I Hz to *per_cell_rates.npz*), *pop_traces* (per-trial population activity to *pop_traces.npz*, base signal for PSD / f_γ), *rasters* (sparse spike indices to *rasters.npz*, for cycle-level analysis).
–tau-gaba FLOAT	inherited / 9.0	[*–infer*] Override τ_GABA (ms) to replay a trained cell under specified inhibitory dynamics. Normally unset: *–load-config* inherits the trained value.
–skip-load PREFIX [dots.c]	—	[*–infer*] Drop *state_dict* keys with these prefixes before loading (e.g. *W_ei. W_ie.*) so a fresh sub-block survives. Transfer-load probes (exp038).
–perturb-mode {drop, add}	—	[*–infer*] Hidden-spike perturbation inside the forward loop: *drop* (Bernoulli mask), *add* (Poisson Hz). The exp037 drop/add asymmetry.
–perturb-level LEVEL [dots.c]	—	[*–perturb-mode*] One value: probability for *drop*, Hz for *add*.

–i-override-file PATH	—	[*–infer*] NPZ with a sparse per-trial I-spike stream to substitute for the inhibitory spikes each timestep. Injection dual of *–outputs rasters* (exp042).
–input-file PATH	—	NPZ with *input_spikes* (T, B, N_IN) to forward instead of Poisson input. Arbitrary stimulus (exp048 digit streams).
–scale-w-in / –scale-w-ei / –scale-w-ie FLOAT	1.0	[*–infer*] Multiply loaded input / E→I / I→E weights before the forward pass. Inference-time coupling sweeps without retraining (exp038).
–sample-index INT	—	Raw test-set index for a snapshot, overriding *–digit / –sample*.
–n-in / –n-inh / –n-batch INT	784 / — / 64	[synthetic-spikes] Input channels, inhibitory pool size, Poisson trials averaged.
–w-ei-mean / –w-ie-mean FLOAT	from *–ei-strength*	[synthetic-spikes] Explicit W_EI / W_IE mean (std = $0.1 \cdot$ mean).
–private-w-in	off	[synthetic-spikes] Identity W_in: one input channel per E cell.

The block from *–skip-load* down is the **generic-primitive family**: small, experiment-agnostic hooks (perturb hidden spikes, inject an inhibitory stream, forward an arbitrary input file, scale a weight block at inference). Runners compose these instead of importing model code, and that is what keeps the tool/experiment boundary clean.

train

Surrogate-gradient BPTT training loop. Writes *weights.pth*, *metrics.json*, a per-step *metrics.jsonl*, and *test_predictions.json*.

```
uv run python tools/snn/tool.py train --model ping --dataset mnist \
  --epochs 100 --lr 0.0001 --v-grad-dampen 1000
```

–epochs 0 runs the init snapshot only, useful as a probe.

flag	default	description
–lr FLOAT	0.01	Adam learning rate. Biophysical models (ping) typically need 0.0001; current-based models 0.01.
–epochs INT	0	Number of epochs. 0 = init-snapshot probe only.
–batch-size INT	64	DataLoader batch size.
–max-samples INT	all	Cap dataset to N samples for smoke tests.
–v-grad-dampen FLOAT	80.0	Dampening factor d on the COBA membrane-voltage gradient. PING needs a much larger value (d = *1000* in the paper) to stabilise BPTT through the E↔I loop; COBA trains at the default. Theory in ar006.
–fr-reg-upper-theta FLOAT	0 (off)	Upper-bound target spike count per neuron per trial (θ_u). Adds $s\_u \cdot \Sigma\ relu(\langle z\_i \rangle - \theta\_u)^2$ to the loss. Cramer et al. SHD RSNN: 100.
–fr-reg-upper-strength FLOAT	0	Coefficient s_u on the upper regulariser. Cramer et al.: 0.06.
–tau-gaba FLOAT	9.0 ms	Override τ_GABA. Default = *models.py*'s value (Börgers / Buzsáki-Wang range). exp041 sweeps this across {4.5 … 27} ms while training PING from scratch; the realised gamma frequency f_γ tracks 1/τ_GABA.

dump-weights

Build the network from a config and emit its weight matrices to *weights_dump.npz*: the init state, plus (with *–load-weights*) the trained state. It runs no forward pass.

```
uv run python tools/snn/tool.py dump-weights \
  --load-config runs/foo/config.json \
  --load-weights runs/foo/weights.pth \
  --out-dir runs/foo/dump
```

Keys follow *W_ff_N_init* / *W_ff_N_trained* (feedforward, per layer N) plus the E-I blocks *W_ei* / *W_ie*. This is how a runner recovers the trained readout matrix (W_out = the last *W_ff*) or compares init-vs-trained loop weights (the exp049 pruning analysis) without loading the model in-process. It takes the shared option groups plus *–load-config* / *–load-weights*.

Shared options

These groups are attached to every subcommand. The grouping matches what each subcommand's *–help* prints.

Network

flag	default	description
–model {ping}	*ping*	Architecture. *ping* is the COBANet with E↔I coupling; with *–ei-strength 0* the inhibitory loop is silenced for a no-rhythm control.
–n-hidden INT [INT dots.c]	dataset-dependent	Hidden layer sizes. One integer = single layer; multiple stacks layers. Default for mnist: 1024.
–readout {rate, mem-mean}	*rate*	Output stage. *rate* sums last-hidden spikes and projects linearly at the final timestep. *mem-mean* averages a per-class output-LIF membrane over time (the trained classification readout).
–dales-law / –no-dales-law	on	Enforce Dale's law (non-negative weights) or allow signed weights. *–no-dales-law* is used for balanced-network experiments.
–ei-strength FLOAT	0.5	E-I coupling strength s. Sets W_EI = s and W_IE = s · ratio.
–ei-ratio FLOAT	2.0	W_IE / W_EI.
–w-in-sparsity FLOAT	0.95	Fraction of input weights zeroed at init.
–ei-sparsity FLOAT	0.0	Sparsity of the recurrent E↔I matrices: fraction of entries zeroed, survivors rescaled by 1/(1−s) to preserve expected drive. Use ≈ 1 − K/N for Brunel/Vreeswijk sparse random connectivity.
–exact-k	off	Fixed-fan-in (exact-K) recurrent connectivity: every post cell draws exactly K = round((1−ei_sparsity) · N_pre) inputs. No effect unless *–ei-sparsity* > 0.
–dt FLOAT	0.25	Integration timestep (ms).
–t-ms FLOAT	200	Total trial duration (ms). Metrics are measured over the full trace; runners strip any startup transient in post.
–readout-w-out-scale FLOAT	1.0	Scalar applied to the readout matrix after *build_net*, compensating for low hidden firing rate under *mem-mean*. Train-mode only.

–surrogate-slope FLOAT	1.0	Fast-sigmoid surrogate-gradient slope β. Larger = narrower active window. Cramer et al. use 40 for SHD RSNNs.

The drive family below also lives in the Network group. It exists for the balanced-network (Brunel / van Vreeswijk-Sompolinsky) experiments and the Lyapunov chaos probe; canonical PING runs leave all of it off.

flag	**default**	**description**
–independent-drive RATE G_PER_SPIKE	off	Per-E-cell independent Poisson drive (bypasses W_in): N_E uncorrelated streams at RATE Hz, each spike adding G_PER_SPIKE μS of g_E. Zero cross-cell correlation.
–independent-drive-i RATE G_PER_SPIKE	off	As above, targeting the I population directly. Needed for the full V&S asynchronous-irregular state.
–quenched-drive MEAN STD	off	Per-E-cell DC conductance drawn once from N(MEAN, STD) μS and frozen for the trial. V&S quenched input: no fluctuation, so it cannot pin spike times; the Lyapunov probe then measures autonomous chaos.
–quenched-drive-i MEAN STD	off	Per-I-cell frozen DC conductance.
–lyapunov-eps FLOAT	0 (off)	If > 0 (synthetic-spikes mode), rerun with all membranes ε-perturbed at t=0 and save the divergence $\|\Delta V(t)\|$ to *snapshot.npz*. Its growth rate is the max Lyapunov exponent: positive for the chaotic V&S state, ≈ 0 for cycle-locked PING.

Input

flag	**default**	**description**
–input {synthetic-spikes, dataset}	*synthetic-spikes*	Stimulus regime. *synthetic-spikes* is Poisson at *–input-rate*. *dataset* draws from *–dataset*.
–input-rate FLOAT	25	Baseline input rate (Hz).
–digit INT	0	Dataset class (0–9).

–sample INT	0	Sample index within the class.
–sample-index INT	—	Raw test-set index, overriding *–digit* / *–sample*.
–dataset {mnist}	*mnist*	Dataset. *mnist* is the full 28×28 image encoded to spikes.

Weights (advanced)

flag	default	description
–w-in MEAN [STD]	*0.3 0.06*	Input fan-in init. Single value sets STD = MEAN × 0.1.
–w-ei MEAN STD	from *–ei-strength*	Override the W_EI init.
–w-ie MEAN STD	from *–ei-strength* / *–ei-ratio*	Override the W_IE init.
–w-ii MEAN STD	*0 0*	W_II (I→I) init. Off by default (canonical PING has no I→I). Enable for balanced-network experiments.
–w-ee MEAN STD	*0 0*	W_EE (E→E) init. Off by default. Enable for the full four-coupling balanced network, where recurrent excitation pins the E rate.
–trainable-w-ei	frozen	Promote E→I to gradient-carrying. Asks whether the optimiser will discover the PING-loop weights from scratch.
–trainable-w-ie	frozen	Promote I→E. The exp049 result *gradient descent dismantles PING* comes from flipping *–trainable-w-ei* and *–trainable-w-ie* on simultaneously.

Output

flag	default	description
–out-dir DIR	*temp/pinglab-cli/*	Output directory. The default is scratch (gitignored); runners always pass an explicit path.
–wipe-dir	off	Clear the output directory before the run.

Execution

flag	default	description
–seed INT	—	RNG seed. Seeds Python, NumPy, torch (CPU + CUDA + MPS) before dataset load and model init. Persisted to *config.json*.

–modal	off	Re-dispatch to Modal.com. Artifacts sync back to *–out-dir* after completion.
–modal-gpu {none, T4, L4, A10G, A100, H100}	*T4*	GPU type for Modal runs. *none* runs CPU-only.

–modal costs money. The project default is local; only pass it when explicitly instructed.

Config inheritance

The trick that makes the experiment chain work is *–load-config*. Every *train* run writes a *config.json* alongside its *weights.pth*; a later *sim* or *dump-weights* run inherits from it:

```
uv run python tools/snn/tool.py sim --infer \
  --load-config runs/foo/config.json \
  --load-weights runs/foo/weights.pth \
  --dt 0.5
```

This inherits the model, hidden sizes, dataset, E-I parameters, input rate, τ_GABA, and seed, while the explicitly-passed *–dt 0.5* overrides the trained value, replaying the network at a new timestep. Precedence is: explicit CLI flag, then loaded config, then default. The parser builds the set of CLI-explicit flags from *sys.argv* before applying inheritance.

Backwards compatibility: old configs that stored *n_hidden* as a scalar are remapped to the *hidden_sizes* list, legacy model names are aliased with a one-line stderr note, and configs missing *dales_law* trigger a warning to pass it explicitly or retrain.

Artifacts

Every subcommand calls *save_run_artifacts* on entry, writing four provenance files into *–out-dir*:

file	**contents**
config.json	The parsed argparse namespace plus a provenance block (*git_sha* with a *dirty* suffix, *torch_version*, *device*, *python_env_hash*, *run_id*, *started_at*) and the *mode*. Consumed by *–load-config* and the runner's metadata extractors.
run.sh	The literal *sys.argv* joined with spaces and prefixed with a shebang. Re-running reproduces the run (modulo seeded RNG state, also in *config.json*).
output.log	The human run log. ANSI escapes are stripped from the file but preserved on stdout, so terminals see colour while the log stays grep-friendly.
run.jsonl	The machine-readable event spine: one typed JSON object per event (epoch rows, warnings, summary). This is what a runner parses when it wants structured progress, not scraped log text.

Each command adds its own outputs: *train* writes *weights.pth*, *metrics.json*, *metrics.jsonl*, *test_predictions.json*; *sim –infer* writes *metrics.json* (and *results.json*) plus whatever *–outputs* requested; *dump-weights* writes *weights_dump.npz*.

These files are scratch. By default they land under *temp/pinglab-cli/*, which is gitignored and overwritten every run. The committed record is produced by the runner: it aggregates each command's config plus headline metrics into one *artifacts/data/expNNN/numbers.json* and renders its figures alongside. That folder, not the ephemeral tool output, is what the publisher reads and what reaches the site.

Recipes

Train, then measure the trained network. Train writes a run directory; *sim –infer* reads it back and emits the population traces a runner needs for a PSD:

```
uv run python tools/snn/tool.py train --dataset mnist --epochs 100 \
  --lr 0.0001 --v-grad-dampen 1000 --out-dir runs/ping

uv run python tools/snn/tool.py sim --infer \
  --load-config runs/ping/config.json \
  --load-weights runs/ping/weights.pth \
  --outputs pop_traces per_cell_rates
```

Loop-transfer at inference (exp038). Take a network trained as COBA and scale its E→I coupling up at inference, with no retraining:

```
uv run python tools/snn/tool.py sim --infer \
  --load-config runs/coba/config.json \
  --load-weights runs/coba/weights.pth \
  --scale-w-ei 1.0 --scale-w-ie 1.0 --outputs rasters
```

Perturbation sweep (exp037). Drop a fraction of emitted spikes, or add off-phase Poisson noise, inside the forward loop:

```
uv run python tools/snn/tool.py sim --infer \
  --load-config runs/ping/config.json --load-weights runs/ping/weights.pth \
  --perturb-mode drop --perturb-level 0.8
```

Recover the trained readout matrix. Dump weights and read *W_ff_N_trained* (the last layer is W_out):

```
uv run python tools/snn/tool.py dump-weights \
  --load-config runs/ping/config.json \
  --load-weights runs/ping/weights.pth --out-dir runs/ping/dump
```