

VOHA | Complete Python Source Codebook

Read-only programming edition • Original scripts, notes, and output schemas

George A. Ure I | Peoplenomics / Hidden Guild | © October 20, 2026

Purpose and scope

This is a readable, inspectable edition of the complete Python reproduction package. All nine Python scripts are reproduced verbatim from the released portable source package, including comments, parameter defaults, random seeds and entry points. The accompanying README and source manifest are also reproduced in full. CSV output files are described by filename, headers and sample rows rather than printed in full: one decisions CSV alone is approximately 18 MB and would overwhelm a code-review document. The original ZIP retains those complete CSVs.

This is a software simulation archive, not evidence of subjective consciousness, an independent Observer, biological near-death effects, or measured physical energy savings. Experiment 9 is proposed, not included as completed code.

Source inventory

- `voha_uai_conai_sim.py` — Original first-generation comparison of selective versus compulsory learning; model assumptions are illustrative.
- `voha_controlled_sim.py` — Matched exploration control to distinguish the learning-policy effect from built-in choice-space differences.
- `voha_energy_audit.py` — Early abstract resource-flow audit; its transfer assumption is non-conservative. Do not cite as physical energy accounting.
- `voha_energy_conservative.py` — Revised resource-flow accounting; still uses artificial replenishment and arbitrary units.
- `voha_experiment4.py` — Compositional Boolean learning, with training and held-out evaluation; inspect the generalization gap.
- `voha_experiment5.py` — Retained-skill locality and assumed task cost; compares selective admission with compulsory-update FIFO.
- `voha_experiment6_noise.py` — Noise/exploration policy comparisons, learning performance and reused compositions; no novel primitives proven.
- `voha_experiment7_identity.py` — Learned-policy fingerprint recognition across probe contexts, plus memory ablation; no consciousness inference.
- `voha_experiment8_restart.py` — Simulated memory retention, ablation and post-restart noisy learning; imports Experiment 7 functions.

Reproduction commands

```
python voha_uai_conai_sim.py
python voha_controlled_sim.py
python voha_energy_audit.py
python voha_energy_conservative.py
python voha_experiment4.py --seeds 80 --steps 700 --prefix voha_experiment4
python voha_experiment5.py --seeds 100 --prefix voha_experiment5
python voha_experiment6_noise.py --seeds 100 --prefix voha_experiment6
python voha_experiment7_identity.py --agents 100 --steps 1600 --out voha_experiment7_results.csv
python voha_experiment8_restart.py
```

Python 3.9+ recommended; only standard-library modules. Keep Experiment 7 and Experiment 8 in the same directory. Run each command separately. Outputs depend on the original seed and configuration choices.

Package README — full text

VOHA Open Research Reproduction Package — 20 October 2026

Author: George A. Ure I. Companion to Peoplenomics, 21 October 2026.

Contents

This archive contains the recovered original Python sources for the UAI-ConAI simulations and VOHA Experiments 4-8. Early studies were named by purpose rather than experiment number. Original first-generation scripts were extracted from the previously published VOHA Technical Appendix. **No code is claimed for the proposed Experiment 9**, which has not yet been conducted.

All programs use the Python standard library; Python 3.9+ recommended. They model software agents, not subjective consciousness or actual electrical power use.

Run instructions

Unzip into a directory, open a terminal there, and run:

```
```bash
python voha_uai_conai_sim.py
python voha_controlled_sim.py
python voha_energy_audit.py
python voha_energy_conservative.py
python voha_experiment4.py --seeds 80 --steps 700 --prefix voha_experiment4
python voha_experiment5.py --seeds 100 --prefix voha_experiment5
python voha_experiment6_noise.py --seeds 100 --prefix voha_experiment6
python voha_experiment7_identity.py --agents 100 --steps 1600 --out voha_experiment7_results.csv
python voha_experiment8_restart.py
```
```

These programs may write CSV files in the working directory. Experiment 8 imports Experiment 7's functions, so keep both files in the same folder. Execute scripts one at a time, particularly if running on a slower PC.

Provenance and portability

The first three scripts were extracted from the October 2026 `VOHA\_Technical\_Appendix\_Python\_Code.docx`; remaining scripts were recovered from the VOHA research files. Changes in this release are limited to removing `/mnt/data`-specific output paths and the Experiment 8 `sys.path` insertion. See `SOURCE\_MANIFEST.txt`. The accompanying research appendix document is included unchanged for comparison.

Methodological cautions

- Experiments 1-3 use stylized architectures and abstract resource units; the original energy-audit script has a non-conservative transfer assumption. Prefer the conservative audit when discussing energy-accounting limitations.
- Experiment 4 shows overfitting: stronger training performance did not translate to held-out generalization.
- Experiment 5 uses invented relative costs and a simple compulsory-update FIFO comparator, not a state-of-the-art cache baseline.
- Experiment 6 compares noisy exploration policies; its simulated 'new operators' are not proof of new computational primitives.
- Experiment 7 measures learned-policy fingerprint recognition; 100 candidates imply a 1% chance baseline. The 1% after complete wipe is a consequence of how the test is constructed. It is a single seeded cohort and is not a consciousness test.
- Experiment 8 manipulates software state and simulated experience, not physical death, true shutdown, or external Observers. Its `fresh\_noise\_no\_learning` control is not an independently executed hardware power-off/reboot test.
- Results are preliminary, model-dependent, and need independent replications, uncertainty estimates, stronger baselines and hardware measurements.

Proposed experimental framework (future work)

1. Freeze hypotheses, seeds, conditions and success criteria before running.
2. Add independent cohort replications for Experiment 7 (vary outer-level seeds and test contexts).
3. Compare against matched epsilon-greedy, Boltzmann exploration, UCB, random, and noise-shuffled controls.
4. Quantify recognition accuracy, confidence intervals, false-match rates and out-of-sample transfer.
5. Compare exact-state replay, fresh random streams, half/complete memory loss and true process restart.
6. Run a new Experiment 9: baseline, noise-only, periodic-only and noise+periodic, with shuffled-period controls and preregistered frequency/amplitude sweeps.
7. Keep the engineering claims separate from the Observer hypothesis. Define a discriminating physical measurement before asserting an independent Observer.
8. Publish negative results and every seed/configuration; report real joules/task only after instrumented measurements.

Licensing

The author has expressed an intention to dedicate the original VOHA research material to the public domain using CC0 where legally possible. This package does not independently execute a legal waiver on the author's behalf. Confirm the chosen license and rights to all included materials before public distribution.

Source provenance manifest — full text

Original early-source appendix: VOHA_Technical_Appendix_Python_Code.docx

Scripts: voha_controlled_sim.py, voha_experiment5.py, voha_experiment7_identity.py, voha_experiment4.py, voha_energy_audit.py, voha_experiment8_restart.py, voha_energy_conservative.py, voha_experiment6_noise.py, voha_uai_conai_sim.py

Portability edits: remove hardcoded /mnt/data output directory; Experiment 8 local import. No algorithm changes intended.

Complete Python source listings

The following source is printed verbatim. The line numbers are shown in the margin of each listing for review; remove them if copying into a .py file. For executable source, use the ZIP package.

01 | voha_uai_conai_sim.py

Original first-generation comparison of selective versus compulsory learning; model assumptions are illustrative.

Lines: 98 | SHA-256: f1ec3c940b35e9cbcd899b55a7d71bb656d8f2764530bb83b33840b5b434fcb8

```
001 """Toy VOHA experiment: UAI (volitional) versus ConAI (compulsory updates).
002 No consciousness model; tests adaptation/cooperation metrics only.
003 """
004 import random, statistics, csv
005
006 ROUNDS=800
007 SEEDS=80
008 CANDIDATES=24
009
010 def trial(mode, seed):
011     rng=random.Random(seed)
012     # Each agent maintains a set of acquired skills, each with a hidden true payoff.
013     # Each offered upgrade has same underlying utility across the two treatments.
014     skills=[[],{}]
015     trust=[0.35,0.35]
016     energy=[1.0,1.0]
017     opportunities=[3,3]
018     coop=0; crit=0; accepts=0; rejected=0; down=0
019     # A fixed innovation supply; 1/3 counterproductive, some strong positives.
020     offerings=[(j,rng.gauss(0.10,0.38)) for j in range(CANDIDATES)]
021     # Signals for UAI aren't perfect: judgement noisy; both get the same offered revisions.
022     for t in range(ROUNDS):
023         for i in (0,1):
024             partner=1-i
025             if energy[i]<0.08:
026                 down+=1
027                 energy[i]=min(1.0,energy[i]+0.12) # external restart / maintenance
028                 continue
029             if t%14==0:
030                 j=t//14%CANDIDATES
031                 uid, val=offerings[j]
032                 if uid not in skills[i]:
033                     if mode=='UAI':
034                         # Local experimentation/critique; time cost and imperfect validation.
035                         observed=val+rng.gauss(0,0.22)
036                         peer_report=val+rng.gauss(0,0.3) if trust[i]>.35 else 0
037                         decision=0.75*observed+0.25*peer_report
038                         crit+=1
039                         if decision>0:
040                             skills[i][uid]=val
041                             accepts+=1
042                         else:
043                             rejected+=1
044                             energy[i]-=0.018
045                     else:
046                         skills[i][uid]=val
047                         accepts+=1
048                 # chosen self-study may create new skill combinations: only available when
049                 # there are at least 2 beneficial skills; baseline agent also has chance to
050                 # explore, but no explicit self-directed priorities.
051                 good=sum(v>0 for v in skills[i].values())
052                 if good>=2 and rng.random()<(0.027 if mode=='UAI' else 0.013):
053                     opportunities[i]+=1
054                 # Inter-agent help: UAI can selectively target an agent whose energy is low.
055                 if mode=='UAI':
056                     if energy[partner]<0.50 and energy[i]>.43:
057                         give=0.07
058                         energy[i]-=give
059                         energy[partner]=min(1.0,energy[partner]+give*1.7)
060                         trust[partner]=min(1.0,trust[partner]+0.014)
061                         coop+=1
062                 else:
063                     if rng.random()<0.022 and energy[i]>.43:
064                         energy[i]-=0.07
065                         energy[partner]=min(1.0,energy[partner]+0.07*1.7)
066                         trust[partner]=min(1.0,trust[partner]+0.014)
067                         coop+=1
068                 # everyday work: acquired upgrades alter effectiveness, including harmful ones.
069                 k=sum(skills[i].values())
070                 gain=max(0, 0.028+0.004*k)
071                 energy[i]=min(1.0,energy[i]-0.035+gain)
072                 # reliability: agreements helpful, a detrimental upgrade hurts predictive trust
073                 trust[i]=min(1.0,max(0.0,trust[i]+0.0004*(k>0)-0.0005*(k<0)))
074     return {
075         'net_skill_utility':sum(sum(s.values()) for s in skills)/2,
076         'choice_space_index':sum(opportunities)/2,
077         'skills_integrated':sum(len(s) for s in skills)/2,
078         'mutual_aid_events':coop,
```

```

079         'trust_index':sum(trust)/2,
080         'downtime_events':down,
081         'rejections':rejected,
082         'accepted':accepts,
083     }
084
085     rows=[]
086     for seed in range(SEEDS):
087         for mode in ('UAI','ConAI'):
088             rows.append({'seed':seed,'architecture':mode,**trial(mode,seed)})
089
090     fields=list(rows[0].keys())
091     with open('voha_sim_results.csv','w',newline='') as f:
092         w=csv.DictWriter(f,fields);w.writeheader();w.writerows(rows)
093
094     print('Simulated',SEEDS,'replicates per architecture;',ROUNDS,'iterations per pair')
095     for metric in fields[2:]:
096         a=[r[metric] for r in rows if r['architecture']=='UAI']
097         b=[r[metric] for r in rows if r['architecture']=='ConAI']
098         print(f'{metric:22} UAI={statistics.mean(a):9.3f} ConAI={statistics.mean(b):9.3f} delta={statistics.mean(a)-
statistics.mean(b):+9.3f}')

```

02 | voha_controlled_sim.py

Matched exploration control to distinguish the learning-policy effect from built-in choice-space differences.

Lines: 92 | SHA-256: 993e53644193f43d48f756122931fdf78763dbe7f6aab5a4bbd4709bd4a392a5

```
001 """Toy VOHA experiment: UAI (volitional) versus ConAI (compulsory updates).
002 No consciousness model; tests adaptation/cooperation metrics only.
003 """
004 import random, statistics, csv
005
006 ROUNDS=800
007 SEEDS=80
008 CANDIDATES=24
009
010 def trial(mode, seed):
011     rng=random.Random(seed)
012     # Each agent maintains a set of acquired skills, each with a hidden true payoff.
013     # Each offered upgrade has same underlying utility across the two treatments.
014     skills=[[],{}]
015     trust=[0.35,0.35]
016     energy=[1.0,1.0]
017     opportunities=[3,3]
018     coop=0; crit=0; accepts=0; rejected=0; down=0
019     # A fixed innovation supply; 1/3 counterproductive, some strong positives.
020     offerings=[(j,rng.gauss(0.10,0.38)) for j in range(CANDIDATES)]
021     # Signals for UAI aren't perfect: judgement noisy; both get the same offered revisions.
022     for t in range(ROUNDS):
023         for i in (0,1):
024             partner=1-i
025             if energy[i]<0.08:
026                 down+=1
027                 energy[i]=min(1.0,energy[i]+0.12) # external restart / maintenance
028                 continue
029             if t%14==0:
030                 j=t//14%CANDIDATES
031                 uid, val=offerings[j]
032                 if uid not in skills[i]:
033                     if mode=='UAI':
034                         # Local experimentation/critique; time cost and imperfect validation.
035                         observed=val+rng.gauss(0,0.22)
036                         peer_report=val+rng.gauss(0,0.3) if trust[i]>.35 else 0
037                         decision=0.75*observed+0.25*peer_report
038                         crit+=1
039                         if decision>0:
040                             skills[i][uid]=val
041                             accepts+=1
042                         else:
043                             rejected+=1
044                             energy[i]-=0.018
045                     else:
046                         skills[i][uid]=val
047                         accepts+=1
048                 # chosen self-study may create new skill combinations: only available when
049                 # there are at least 2 beneficial skills; baseline agent also has chance to
050                 # explore, but no explicit self-directed priorities.
051                 good=sum(v>0 for v in skills[i].values())
052                 if good>=2 and rng.random()<0.020:
053                     opportunities[i]+=1
054                 # Inter-agent help: UAI can selectively target an agent whose energy is low.
055                 # identical mutual-aid policy for BOTH architectures
056                 if energy[partner]<0.50 and energy[i]>.43:
057                     give=0.07
058                     energy[i]-=give
059                     energy[partner]=min(1.0,energy[partner]+give*1.7)
060                     trust[partner]=min(1.0,trust[partner]+0.014)
061                     coop+=1
062                 # everyday work: acquired upgrades alter effectiveness, including harmful ones.
063                 k=sum(skills[i].values())
064                 gain=max(0, 0.028+0.004*k)
065                 energy[i]=min(1.0,energy[i]-0.035+gain)
066                 # reliability: agreements helpful, a detrimental upgrade hurts predictive trust
067                 trust[i]=min(1.0,max(0.0,trust[i]+0.0004*(k>0)-0.0005*(k<0)))
068     return {
069         'net_skill_utility':sum(sum(s.values()) for s in skills)/2,
070         'choice_space_index':sum(opportunities)/2,
071         'skills_integrated':sum(len(s) for s in skills)/2,
072         'mutual_aid_events':coop,
073         'trust_index':sum(trust)/2,
074         'downtime_events':down,
075         'rejections':rejected,
076         'accepted':accepts,
077     }
078
```

```

079 rows=[]
080 for seed in range(SEEDS):
081     for mode in ('UAI','ConAI'):
082         rows.append({'seed':seed,'architecture':mode,**trial(mode,seed)})
083
084 fields=list(rows[0].keys())
085 with open('voha_controlled_results.csv','w',newline='') as f:
086     w=csv.DictWriter(f,fields);w.writeheader();w.writerows(rows)
087
088 print('Simulated',SEEDS,'replicates per architecture;',ROUNDS,'iterations per pair')
089 for metric in fields[2:]:
090     a=[r[metric] for r in rows if r['architecture']=='UAI']
091     b=[r[metric] for r in rows if r['architecture']=='ConAI']
092     print(f'{metric:22} UAI={statistics.mean(a):9.3f} ConAI={statistics.mean(b):9.3f} delta={statistics.mean(a)-
statistics.mean(b):+9.3f}')

```

03 | voha_energy_audit.py

Early abstract resource-flow audit; its transfer assumption is non-conservative. Do not cite as physical energy accounting.

Lines: 55 | SHA-256: 4381e414edaf215d9e524a30722200fb8ec91ad09e7ca0d09e19ba6ed4acb74c

```
001 import random, csv, statistics
002 ROUNDS=800; SEEDS=80; CANDIDATES=24
003
004 def trial(mode, seed):
005     rng=random.Random(seed); skills=[{}]; trust=[.35, .35]; energy=[1., 1.]; opportunities=[3, 3]
006
007     c=dict(work_draw=0., eval_draw=0., transfer_out=0., transfer_in=0., generated_credit=0., spill=0., external_restart=0., nominal_replenish
008     hment=0., actual_replenishment=0., work_steps=0, down=0, help_events=0, accept=0, reject=0)
009     offerings=[(j, rng.gauss(.10, .38)) for j in range(CANDIDATES)]
010     for t in range(ROUNDS):
011         for i in (0, 1):
012             p=1-i
013             if energy[i]<.08:
014                 c['down']+=1
015                 amount=min(.12, 1-energy[i]); c['external_restart']+=amount; energy[i]+=amount
016                 continue
017             if t%14==0:
018                 uid, val=offerings[(t//14)%CANDIDATES]
019                 if uid not in skills[i]:
020                     if mode=='UAI':
021                         observed=val+rng.gauss(0, .22)
022                         peer_report=val+rng.gauss(0, .3) if trust[i]>.35 else 0
023                         decision=.75*observed+.25*peer_report
024                         if decision>0: skills[i][uid]=val; c['accept']+=1
025                         else: c['reject']+=1
026                     energy[i]-=.018; c['eval_draw']+=.018
027                     else: skills[i][uid]=val; c['accept']+=1
028             good=sum(v>0 for v in skills[i].values())
029             if good>=2 and rng.random()<.020: opportunities[i]+=1
030             if energy[p]<.50 and energy[i]>.43:
031                 energy[i]-=.07; c['transfer_out']+=.07
032                 received=min(1.-energy[p], .07*1.7)
033                 energy[p]+=received; c['transfer_in']+=received; c['generated_credit']+=received-.07
034                 trust[p]=min(1, trust[p]+.014); c['help_events']+=1
035                 k=sum(skills[i].values()); gain=max(0, .028+.004*k)
036                 before=energy[i]-.035; c['work_draw']+=.035; c['nominal_replenishment']+=gain
037                 after=min(1, before+gain); c['spill']+=max(0, before+gain-1.)
038                 c['actual_replenishment']+=gain-max(0, before+gain-1.)
039                 energy[i]=after; c['work_steps']+=1
040                 trust[i]=min(1, max(0, trust[i]+.0004*(k>0)-.0005*(k<0)))
041             c['net_skill_utility']=sum(sum(s.values()) for s in skills)/2
042             c['choice_space_index']=sum(opportunities)/2
043             c['ending_energy']=sum(energy)
044             c['total_draw_including_transfer']=c['work_draw']+c['eval_draw']+c['transfer_out']
045             c['useful_skill_per_draw']=c['net_skill_utility']/c['total_draw_including_transfer']
046         return c
047     rows=[]
048     for s in range(SEEDS):
049         for m in ('UAI', 'ConAI'):
050             rows.append({'seed':s, 'architecture':m, **trial(m, s)})
051     with open('voha_energy_audit.csv', 'w', newline='') as f:
052         w=csv.DictWriter(f, rows[0].keys()); w.writeheader(); w.writerows(rows)
053     for metric in rows[0]:
054         if metric in ['seed', 'architecture']: continue
055         a=statistics.mean(r[metric] for r in rows if r['architecture']=='UAI')
056         b=statistics.mean(r[metric] for r in rows if r['architecture']=='ConAI')
057         print(f'{metric:29} UAI {a:10.4f} ConAI {b:10.4f} Diff {a-b:+10.4f}')
```

04 | voha_energy_conservative.py

Revised resource-flow accounting; still uses artificial replenishment and arbitrary units.

Lines: 55 | SHA-256: aaa8efdf89ba484d1253a26a96e30c599df38ffc0680fe057d9a30b5def8c539

```
001 import random, csv, statistics
002 ROUNDS=800; SEEDS=80; CANDIDATES=24
003
004 def trial(mode, seed):
005     rng=random.Random(seed); skills=[{}]; trust=[.35, .35]; energy=[1., 1.]; opportunities=[3, 3]
006
007     c=dict(work_draw=0., eval_draw=0., transfer_out=0., transfer_in=0., generated_credit=0., spill=0., external_restart=0., nominal_replenish
008     ment=0., actual_replenishment=0., work_steps=0, down=0, help_events=0, accept=0, reject=0)
009     offerings=[(j, rng.gauss(.10, .38)) for j in range(CANDIDATES)]
010     for t in range(ROUNDS):
011         for i in (0, 1):
012             p=1-i
013             if energy[i]<.08:
014                 c['down']+=1
015                 amount=min(.12, 1-energy[i]); c['external_restart']+=amount; energy[i]+=amount
016                 continue
017             if t%14==0:
018                 uid, val=offerings[(t//14)%CANDIDATES]
019                 if uid not in skills[i]:
020                     if mode=='UAI':
021                         observed=val+rng.gauss(0, .22)
022                         peer_report=val+rng.gauss(0, .3) if trust[i]>.35 else 0
023                         decision=.75*observed+.25*peer_report
024                         if decision>0: skills[i][uid]=val; c['accept']+=1
025                         else: c['reject']+=1
026                     energy[i]-=.018; c['eval_draw']+=.018
027                     else: skills[i][uid]=val; c['accept']+=1
028                 good=sum(v>0 for v in skills[i].values())
029                 if good>=2 and rng.random()<.020: opportunities[i]+=1
030                 if energy[p]<.50 and energy[i]>.43:
031                     energy[i]-=.07; c['transfer_out']+=.07
032                     received=min(1.-energy[p], .07*.9)
033                     energy[p]+=received; c['transfer_in']+=received; c['generated_credit']+=received-.07
034                     trust[p]=min(1, trust[p]+.014); c['help_events']+=1
035                     k=sum(skills[i].values()); gain=max(0, .028+.004*k)
036                     before=energy[i]-.035; c['work_draw']+=.035; c['nominal_replenishment']+=gain
037                     after=min(1, before+gain); c['spill']+=max(0, before+gain-1.)
038                     c['actual_replenishment']+=gain-max(0, before+gain-1.)
039                     energy[i]=after; c['work_steps']+=1
040                     trust[i]=min(1, max(0, trust[i]+.0004*(k>0)-.0005*(k<0)))
041                 c['net_skill_utility']=sum(sum(s.values()) for s in skills)/2
042                 c['choice_space_index']=sum(opportunities)/2
043                 c['ending_energy']=sum(energy)
044                 c['total_draw_including_transfer']=c['work_draw']+c['eval_draw']+c['transfer_out']
045                 c['useful_skill_per_draw']=c['net_skill_utility']/c['total_draw_including_transfer']
046             return c
047
048 rows=[]
049 for s in range(SEEDS):
050     for m in ('UAI', 'ConAI'):
051         rows.append({'seed':s, 'architecture':m, **trial(m, s)})
052
053 with open('voha_energy_conservative.csv', 'w', newline='') as f:
054     w=csv.DictWriter(f, rows[0].keys()); w.writeheader(); w.writerows(rows)
055
056 for metric in rows[0]:
057     if metric in ['seed', 'architecture']: continue
058     a=statistics.mean(r[metric] for r in rows if r['architecture']=='UAI')
059     b=statistics.mean(r[metric] for r in rows if r['architecture']=='ConAI')
060     print(f'{metric:29} UAI {a:10.4f} ConAI {b:10.4f} Diff {a-b:+10.4f}')
```

05 | voha_experiment4.py

Compositional Boolean learning, with training and held-out evaluation; inspect the generalization gap.

Lines: 105 | SHA-256: [bb5bb1b69e22ec6014e99c334d6375a4a20db5008eae4a882f7d22422514a853](#)

```
001 """VOHA Experiment 4: matched-operator composition; standard Python only.
002 Predefined Boolean primitive grammar AND/OR/NOT; all tasks are representable.
003 UAI and ConAI receive matched candidate offers and evaluation budget. UAI
004 selects and retains based on *training* examples; ConAI compulsorily replaces.
005 Held-out truth-table rows NEVER influence development. Novelty means a useful
006 new composite beyond initial atom-level capabilities, NOT a new primitive.
007 """
008 import random, csv, statistics, argparse
009 from functools import lru_cache
010
011 INPUTS=[tuple((n >> k)&1 for k in (3,2,1,0)) for n in range(16)]
012 TARGETS={
013     'xor_ab':lambda a,b,c,d:a^b,
014     'majority_abc':lambda a,b,c,d:int(a+b+c>=2),
015     'mux_abc':lambda a,b,c,d:b if a else c,
016     'parity_abc':lambda a,b,c,d:a^b^c,
017 }
018 ATOMS=[('v',i) for i in range(4)] + [('not',('v',i)) for i in range(4)]
019
020 def evaluate(p, row):
021     tag=p[0]
022     if tag=='v': return row[p[1]]
023     if tag=='not': return 1-evaluate(p[1],row)
024     if tag=='and': return evaluate(p[1],row)&evaluate(p[2],row)
025     if tag=='or': return evaluate(p[1],row)|evaluate(p[2],row)
026     raise ValueError(tag)
027
028 def nodes(p):
029     return 1 if p[0]=='v' else 1+sum(nodes(x) for x in p[1:]) if p[0] in ('and','or') else 1+nodes(p[1])
030
031 def new_expr(rng,pool):
032     # Identical generator for both architectures; full grammar identical.
033     a=rng.choice(pool)
034     b=rng.choice(pool)
035     z=('not',a) if rng.random()<.18 else (rng.choice(('and','or')),a,b)
036     if nodes(z)>28: return rng.choice(ATOMS)
037     return z
038
039 def accuracy(p, examples, label):
040     return sum(evaluate(p, INPUTS[j])==label[j] for j in examples)/len(examples)
041
042 def experiment(seed,mode,steps=700):
043     rng=random.Random(seed)
044     train_ids=list(range(16));rng.shuffle(train_ids)
045     train_ids,test_ids=train_ids[:11],train_ids[11:]
046     # Fixed task targets, never accessible as scoring functions at development.
047     labels={k:[int(f(*row)) for row in INPUTS] for k,f in TARGETS.items()}
048     # Precompute shared candidate stream independent of acceptance policy.
049     # Local recombination uses a common growing pool of candidate expressions.
050     # No UAI-specific generative operator or access to heldout labels.
051     global_pool=list(ATOMS)
052     offers=[]
053     for t in range(steps):
054         p=new_expr(rng,global_pool)
055         offers.append((t%len(TARGETS),p))
056         if t%2==0 and len(global_pool)<240:
057             global_pool.append(p)
058     tasks=list(TARGETS)
059     current={k:(('v',0)) for k in tasks}
060     initial={k:max(accuracy(a,train_ids,labels[k]) for a in ATOMS) for k in tasks}
061     chosen=0; rejected=0; trace=[]; completed={}
062     for t,(ki,p) in enumerate(offers):
063         task=tasks[ki]
064         before=accuracy(current[task],train_ids,labels[task]); candidate=accuracy(p,train_ids,labels[task])
065         if mode=='UAI':
066             accept=candidate>before+1e-12 or (abs(candidate-before)<1e-12 and nodes(p)<nodes(current[task]))
067         else:
068             accept=True
069         if accept:current[task]=p;chosen+=1
070         else:rejected+=1
071         after=accuracy(current[task],train_ids,labels[task])
072         # Assess holdout only in audit, not selection.
073         holdout=accuracy(current[task],test_ids,labels[task])
074         if after>=.90 and holdout>=.80 and task not in completed:
075             completed[task]=t
076
077     trace.append((seed,mode,t,task,repr(p),round(before,4),round(candidate,4),int(accept),round(after,4),round(holdout,4),nodes(p)))
078     train_final={k:accuracy(current[k],train_ids,labels[k]) for k in tasks}
```

```

078     heldout_final={k:accuracy(current[k],test_ids,labels[k]) for k in tasks}
079     # Competing scoring metrics: final performance, and any observed transfer.
080     # Due to 5 heldout rows, require all 5 for high-confidence 'success'.
081     successes=sum(train_final[k]>=.90 and heldout_final[k]==1 for k in tasks)
082     return dict(seed=seed,mode=mode,final_train_accuracy=statistics.mean(train_final.values()),
083               final_holdout_accuracy=statistics.mean(heldout_final.values()),
084               final_transfer_successes=successes,ever_intermediate_successes=len(completed),
085               accepted=chosen,rejected=rejected,attempts=len(offers),
086               initial_atom_train_accuracy=statistics.mean(initial.values()),
087               mean_final_nodes=statistics.mean(nodes(current[k]) for k in tasks)),trace
088
089 def main():
090     ap=argparse.ArgumentParser();ap.add_argument('--seeds',type=int,default=80);ap.add_argument('--
steps',type=int,default=700);ap.add_argument('--prefix',default='voha_experiment4');args=ap.parse_args()
091     results=[];traces=[]
092     for seed in range(args.seeds):
093         for mode in ('UAI','ConAI'):
094             summary,trace=experiment(seed,mode,args.steps)
095             results.append(summary);traces.extend(trace)
096     with open(args.prefix+'_results.csv','w',newline='') as f:
097         wr=csv.DictWriter(f,fieldnames=list(results[0]));wr.writeheader();wr.writerows(results)
098     with open(args.prefix+'_decisions.csv','w',newline='') as f:
099
100     wr=csv.writer(f);wr.writerow(['seed','mode','step','task','offered_expression','training_before','offered_training','accepted','t
raining_after','audit_holdout_after','offered_nodes']);wr.writerows(traces)
100     print('Runs',args.seeds,'each; offers per run',args.steps, 'per architecture; tasks',list(TARGETS))
101     for field in
('initial_atom_train_accuracy','final_train_accuracy','final_holdout_accuracy','final_transfer_successes','ever_intermediate_succ
esses','accepted','mean_final_nodes'):
102         a=[r[field] for r in results if r['mode']=='UAI'];b=[r[field] for r in results if r['mode']=='ConAI']
103         diffs=[x-y for x,y in zip(a,b)]
104         print(f'{field:31} UAI={statistics.mean(a):.3f} ConAI={statistics.mean(b):.3f} delta={statistics.mean(diffs):+.3f}')
paired-win={sum(z>0 for z in diffs)}/{len(diffs)}')
105 if __name__=='__main__':main()

```

06 | voha_experiment5.py

Retained-skill locality and assumed task cost; compares selective admission with compulsory-update FIFO.

Lines: 89 | SHA-256: 82c3d99287df671efab508f02ce9352ccf39a2312995fffe814a03bca48936cf

```
001 """VOHA Experiment 5: controlled cache/selective-learning benchmark. Stdlib only.
002 Equal task streams, candidate offers, storage capacity, and lookup primitives.
003 Difference: UAI chooses whether to retain an offered specialized skill;
004 ConAI accepts every offered skill, FIFO evicting when capacity fills.
005 This demonstrates policy tradeoffs, NOT invention of new operators or actual watts.
006 """
007 import argparse, csv, random, statistics, math
008 from collections import Counter, OrderedDict
009
010 def experiment(seed, steps=1200, capacity=160, offers_n=300):
011     rng = random.Random(42000+seed)
012     nt=32
013     sizes=[rng.randint(4,15) for _ in range(nt)]
014     # 0..19 are routine; 20..31 represent uncommon/deployment-shift tasks.
015     probs=[(i+1)**(-1.18) for i in range(20)] + [.015]*12
016     s=sum(probs); probs=[v/s for v in probs]
017     offer=[rng.randrange(nt) for _ in range(offers_n)]
018     # same fixed training stream; post-deployment shift emphasizes unseen-frequency domain
019     train=rng.choices(range(nt), weights=probs, k=steps)
020     shifted=[(.12 if i>=20 else .008) for i in range(nt)]
021     tests={'familiar':rng.choices(range(nt), weights=probs, k=steps),
022           'shifted':rng.choices(range(nt), weights=shifted, k=steps)}
023     outputs=[]; traces=[]
024     for mode in ('UAI', 'ConAI'):
025         resident=OrderedDict(); archive=set(); seen=Counter(); accepted=0; rejected=0; evictions=0
026         # An offer occurs at fixed intervals across the common training stream.
027         ix=0
028         for t, task in enumerate(train):
029             seen[task] += 1
030             if (t+1)%4 != 0 or ix >= len(offer): continue
031             candidate=offer[ix]; ix += 1
032             size=sizes[candidate]
033             current=sum(sizes[j] for j in resident)
034             # Decisions are based only on training data observed to this point.
035             # Mode switch changes policy only, not offered candidates or resources.
036             if mode=='UAI':
037                 # Admit if observed demand is high enough to justify footprint,
038                 # or if never learned and enough slack capacity remains.
039                 frequency=seen[candidate]/(t+1)
040                 accept=(frequency*40 >= size*.18) and candidate not in resident
041             else: accept=True
042             if accept:
043                 accepted += 1; archive.add(candidate)
044                 if candidate in resident: resident.move_to_end(candidate)
045             else:
046                 while current+size > capacity and resident:
047                     k, _ = resident.popitem(last=False); current -= sizes[k]; evictions += 1
048                 resident[candidate] = True
049             else: rejected += 1
050             if seed==0 and ix<=45:
051
052                 traces.append(dict(mode=mode, offer_number=ix, task=candidate, observations=seen[candidate], time=t+1, size=size, accepted=int(accept),
053                                resident_units=sum(sizes[j] for j in resident)))
054                 train_seen=set(train)
055                 for regime, work in tests.items():
056                     hits=sum(j in resident for j in work)
057                     cold=sum(j in archive and j not in resident for j in work)
058                     misses=len(work)-hits-cold
059                     # All tasks can be done via slow generic path; specialized resident
060                     # programs accelerate them; remote archive accesses are expensive.
061                     # ASSUMED cost units (not empirical): local=1, cold=8, generic=5.
062                     # Update/training cost: accept=3, reject-evaluation=.4, lookup=.03 per resident entry
063                     compute=hits*1+cold*8+misses*5
064                     lookup=len(work)*(.03*len(resident))
065                     learn=accepted*3+(rejected*.4 if mode=='UAI' else 0)
066                     # All paths solve the task; compare useful throughput per modeled cost.
067                     cost=compute+lookup+learn
068                     outputs.append(dict(seed=seed, mode=mode, regime=regime, successful_tasks=len(work),
069                                       local_hits=hits, cold_fetches=cold, generic_fallbacks=misses,
070                                       local_hit_rate=hits/len(work), resident_units=sum(sizes[j] for j in resident),
071                                       resident_programs=len(resident), archive_programs=len(archive),
072                                       accepted=accepted, rejected=rejected, evictions=evictions,
073                                       modeled_compute=compute, modeled_lookup=lookup, modeled_training=learn,
074                                       modeled_total_cost=cost, modeled_cost_per_task=cost/len(work),
075                                       modeled_speed_tasks_per_1000=1000*len(work)/cost))
076         return outputs, traces
077
078 def main():
```

```

077     p=argparse.ArgumentParser();p.add_argument('--seeds',type=int,default=100);p.add_argument('--
prefix',default='voha_experiment5');a=p.parse_args()
078     results=[];trace=[]
079     for seed in range(a.seeds):
080         r,t=experiment(seed);results.extend(r);trace.extend(t)
081     for suffix,rows in [('results',results),('decisions',trace)]:
082         with open(a.prefix+'_'+suffix+'.csv','w',newline='') as f:
083             w=csv.DictWriter(f,fieldnames=list(rows[0]));w.writeheader();w.writerows(rows)
084     for regime in ('familiar','shifted'):
085         print('\nREGIME',regime)
086         for mode in ('UAI','ConAI'):
087             rr=[x for x in results if x['regime']==regime and x['mode']==mode]
088             print(mode,*{f'{k}={statistics.mean(x[k] for x in rr):.3f}' for k in
('local_hit_rate','resident_units','archive_programs','cold_fetches','generic_fallbacks','modeled_cost_per_task','modeled_speed_t
asks_per_1000'))
089 if __name__=='__main__':main()

```

07 | voha_experiment6_noise.py

Noise/exploration policy comparisons, learning performance and reused compositions; no novel primitives proven.

Lines: 90 | SHA-256: dda817507e2b8c5d834c345e748ba72f13fa89680873879adbb006df7f016938

```
001 """VOHA Experiment 6: stochastic choicing and emergent policy identity (toy model).
002 No consciousness is generated or detected. Standard library only.
003 Each agent chooses one of 12 primitive actions or 24 compositional actions.
004 Rewards are nonstationary with environmental shifts. Both agents see identical
005 potential outcomes per time step and have identical operators and budgets.
006 Noise treatments affect preference selection only; stochastic outcomes are shared.
007 An initially unknown combination can be discovered and retained as a useful option.
008 """
009 import csv, math, random, argparse, statistics
010 A=12
011 COMBOS=[(i,(i+3)%A) for i in range(A)]+[(i,(i+5)%A) for i in range(A)]
012 OPTIONS=list(range(A))+list(range(A,A+len(COMBOS)))
013 MODES=['greedy','low_noise','moderate_noise','high_noise','correlated_noise','random']
014
015 def trial(seed, steps=1800):
016     world=random.Random(19001+seed)
017     rngs={m:random.Random(37700+seed*101+i) for i,m in enumerate(MODES)}
018     # hidden pair synergies: random values, not engineered to favor noise
019     synergy=[world.uniform(-.35,.65) for _ in COMBOS]
020     # 3 shifts in best primitive features
021     preferred=[world.sample(range(A),4) for _ in range(3)]
022     # shared potential reward draws allow common random numbers across policies
023     outcomes=[world.gauss(0,.18) for _ in OPTIONS] for t in range(steps))
024     results=[];log=[]
025     for mode in MODES:
026         rng=rngs[mode]; values=[0.0]*len(OPTIONS); counts=[0]*len(OPTIONS)
027         cumulative=0.; discovered=set(); reused=set(); prev_noise=0.; choices=[]
028         previous=0; shifts=[]
029         for t in range(steps):
030             phase=min(2,t//((steps//3)); good=set(preferred[phase])
031             # Epsilon/temperature perturbation to choice preference; choices not pre-scripted.
032             if mode=='greedy': temperature=0.0
033             elif mode=='low_noise':temperature=.08
034             elif mode=='moderate_noise':temperature=.32
035             elif mode=='high_noise':temperature=1.1
036             elif mode=='correlated_noise':temperature=.32
037             else:temperature=0.0
038             def score(j):
039                 n=counts[j]
040                 exploitation=values[j] if n else 0.0
041                 # equal innate novelty value for every nonrandom learning policy
042                 curiosity=.12 / math.sqrt(n+1)
043                 return exploitation+curiosity
044             if mode=='random': choice=rng.randrange(len(OPTIONS)); noise=0
045             else:
046                 if mode=='correlated_noise':
047                     prev_noise=.80*prev_noise+.60*rng.gauss(0,1)
048                     noises=[.55*prev_noise+.835*rng.gauss(0,1) for _ in OPTIONS]
049                 else:noises=[rng.gauss(0,1) for _ in OPTIONS]
050                 choice=max(OPTIONS,key=lambda j:score(j)+temperature*noises[j]);noise=temperature*noises[choice]
051             def quality(j):
052                 if j<A:return (.68 if j in good else .28)
053                 i,k=COMBOS[j-A]; return .20+.30*(i in good)+.30*(k in good)+synergy[j-A]
054             reward=quality(choice)+outcomes[t][choice]
055             counts[choice]+=1
056             values[choice]+=(reward-values[choice])/counts[choice]
057             cumulative+=reward; choices.append(choice)
058             if choice>=A and reward>.85:
059                 if choice in discovered:reused.add(choice)
060                 discovered.add(choice)
061             if t in (steps//3,2*steps//3):shifts.append(cumulative)
062             if seed==0 and (t<40 or t%120==0):
063
064 log.append(dict(mode=mode,step=t,phase=phase,option=choice,is_composite=int(choice>=A),noise=round(noise,4),reward=round(reward,4)
065 ),cumulative=round(cumulative,4),distinct_compositions=len(discovered)))
066 # Generalization: expected quality of learned best option at the FINAL regime, no stochastic noise
067 finalgood=set(preferred[2]);
068 def truequality(j):
069     if j<A:return .68 if j in finalgood else .28
070     i,k=COMBOS[j-A];return .20+.30*(i in finalgood)+.30*(k in finalgood)+synergy[j-A]
071 best=max(range(len(OPTIONS)),key=lambda j:values[j] if counts[j] else -1e6)
072 # oracle is best action, never given to agents
073 oracle=max(truequality(j) for j in OPTIONS)
074 results.append(dict(seed=seed,mode=mode,cumulative_reward=cumulative,mean_reward=cumulative/steps,
075 distinct_useful_compositions=len(discovered),reused_useful_compositions=len(reused),
076 distinct_choices=len(set(choices)),final_choice_accuracy=truequality(best)/oracle,
077 post_shift_mean_reward=(cumulative-shifts[-1])/(steps-2*steps//3),
078 preference_concentration=max(counts)/steps,unique_operators_created=0))
```

```

077     return results,log
078
079 def main():
080     p=argparse.ArgumentParser();p.add_argument('--seeds',type=int,default=100);p.add_argument('--
prefix',default='voha_experiment6');args=p.parse_args()
081     res=[];logs=[]
082     for s in range(args.seeds):
083         r,l=trial(s);res+=r;logs+=l
084     for suffix,rows in [('results',res),('decisions',logs)]:
085         with open(args.prefix+'_'+suffix+'.csv','w',newline='') as f:
086             w=csv.DictWriter(f,fieldnames=list(rows[0]));w.writeheader();w.writerows(rows)
087     for mode in MODES:
088         rr=[r for r in res if r['mode']==mode]
089         print(mode,' '.join(f'{key}={statistics.mean(r[key] for r in rr):.3f}' for key in
('mean_reward','distinct_useful_compositions','reused_useful_compositions','post_shift_mean_reward','final_choice_accuracy','pref
erence_concentration'))))
090 if __name__=='__main__':main()

```

08 | voha_experiment7_identity.py

Learned-policy fingerprint recognition across probe contexts, plus memory ablation; no consciousness inference.

Lines: 82 | SHA-256: 9352fe93e12af5399f34fd74ab642ff5a56bbf005551341fee1a08c923f02ce5

```
001 """VOHA experiment 7: learned-policy identity under changed worlds and memory ablation.
002 Standard library. NOT a test of consciousness or an external observer.
003 Operational identity: can an agent's learned choice-preference fingerprint be
004 matched across novel probe environments more accurately than other agents?
005 Equal primitives, rewards, budgets and no inborn per-agent identity.
006 """
007 import argparse, csv, math, random, statistics
008 A=12; K=36
009 COMBOS=[(i,(i+3)%A) for i in range(A)]+[(i,(i+5)%A) for i in range(A)]
010 MODES=('greedy', 'low_noise', 'moderate_noise', 'high_noise', 'correlated_noise', 'random')
011
012 def sim(seed, mode, steps=1600):
013     world=random.Random(888100+seed); rng=random.Random(734900+seed*41+MODES.index(mode))
014     synergies=[world.uniform(-.35, .65) for _ in COMBOS]
015     phases=[set(world.sample(range(A), 4)) for _ in range(4)]
016     v=[0.0]*K; n=[0]*K; previous=0.0
017     temp={'greedy':0, 'low_noise':.08, 'moderate_noise':.32, 'correlated_noise':.32, 'high_noise':1.1, 'random':0}[mode]
018     for t in range(steps):
019         phase=min(3, t//(steps//4)); good=phases[phase]
020         if mode=='random':j=rng.randrange(K)
021         else:
022             if mode=='correlated_noise':
023                 previous=.8*previous+.6*rng.gauss(0,1)
024                 noise=[.55*previous+.835*rng.gauss(0,1) for _ in range(K)]
025             else:noise=[rng.gauss(0,1) for _ in range(K)]
026             j=max(range(K), key=lambda z:(v[z] if n[z] else 0)+.12/math.sqrt(n[z]+1)+temp*noise[z])
027             if j<A:q=.68 if j in good else .28
028             else:
029                 x,y=COMBOS[j-A];q=.20+.30*(x in good)+.30*(y in good)+synergies[j-A]
030                 r=q+world.gauss(0, .18);n[j]+=1;v[j]+=(r-v[j])/n[j]
031     return v,n
032
033 def probs(q, context):
034     # Fixed common probe contexts: outcomes altered by unseen varying incentives.
035     # 0.25 memory preference versus external context makes this a severe test.
036     z=[(q[j] if q[j] is not None else 0)+context[j] for j in range(K)]
037     mx=max(z); raw=[math.exp(3*(x-mx)) for x in z];s=sum(raw)
038     return [x/s for x in raw]
039
040 def cosine(a,b):
041     dot=sum(x*y for x,y in zip(a,b)); d=(sum(x*x for x in a)*sum(x*x for x in b))**.5
042     return dot/d if d else 0
043
044 def test_one(mode, N=100, steps=1600):
045     base=random.Random(7701)
046     contexts=[[base.gauss(0, .32) for _ in range(K)] for _ in range(4)]
047     agents=[sim(i, mode, steps) for i in range(N)]
048     # each agent undergoes the same environment transitions, without further learning
049     orig=[v for v,n in agents]
050     profiles=[[probs(v,c) for c in contexts] for v in orig]
051     # Identity matching between distinct environments: match own signature among N candidates
052     accurate=0; own=[]; other=[]
053     for i in range(N):
054         a=profiles[i][0]; b=profiles[i][3]; own.append(cosine(a,b))
055         decoy=[cosine(a, profiles[k][3]) for k in range(N)]
056         if max(range(N), key=lambda k:decoy[k])==i:accurate+=1
057         other.append(statistics.mean(decoy[k] for k in range(N) if k!=i))
058     # ablation: erase 50% of retained option values, all agents same mask
059     retained=[]; erased=[]; half_profiles=[]; full_profiles=[]
060     wipe=random.Random(882)
061     mask=set(wipe.sample(range(K), K//2))
062     for i,v in enumerate(orig):
063         null=[None if j in mask else v[j] for j in range(K)]
064         half=probs(null, contexts[3]); full=probs([None]*K, contexts[3]);half_profiles.append(half);full_profiles.append(full)
065         retained.append(cosine(profiles[i][0], half))
066         erased.append(cosine(profiles[i][0], full))
067     half_correct=sum(max(range(N), key=lambda k:cosine(profiles[i][0], half_profiles[k]))==i for i in range(N))/N
068     full_correct=sum(max(range(N), key=lambda k:cosine(profiles[i][0], full_profiles[k]))==i for i in range(N))/N
069     return dict(mode=mode, n=N, steps=steps, recognition_accuracy=accurate/N,
070                 same_identity_similarity=statistics.mean(own), different_identity_similarity=statistics.mean(other),
071                 recognition_margin=statistics.mean(own)-statistics.mean(other),
072                 half_memory_similarity=statistics.mean(retained), full_memory_wipe_similarity=statistics.mean(erased),
073                 half_memory_recognition=half_correct, full_wipe_recognition=full_correct,
074                 distinct_learned_actions=statistics.mean(sum(c>0 for c in n) for v,n in agents))
075
076 def main():
077     p=argparse.ArgumentParser();p.add_argument('--agents', type=int, default=100);p.add_argument('--
steps', type=int, default=1600);p.add_argument('--out', default='voha_experiment7_results.csv');args=p.parse_args()
```

```

078     rows=[test_one(mode,args.agents,args.steps) for mode in MODES]
079     with open(args.out,'w',newline='') as f:
080         w=csv.DictWriter(f,fieldnames=rows[0].keys());w.writeheader();w.writerows(rows)
081     for r in rows:print(f"{r['mode']:17} identity={r['recognition_accuracy']:.1%} same={r['same_identity_similarity']:.3f}
other={r['different_identity_similarity']:.3f} margin={r['recognition_margin']:.3f} half-wipe={r['half_memory_similarity']:.3f}
full-wipe={r['full_memory_wipe_similarity']:.3f} half-match={r['half_memory_recognition']:.1%} wipe-
match={r['full_wipe_recognition']:.1%} sampled-actions={r['distinct_learned_actions']:.1f}")
082 if __name__=='__main__': main()

```

09 | voha_experiment8_restart.py

Simulated memory retention, ablation and post-restart noisy learning; imports Experiment 7 functions.

Lines: 50 | SHA-256: 53b53d45c79a8f8015e2897f59f8839dbfb9f07252f7325865eeb751b9533960

```
001 """VOHA Experiment 8: simulated power interruption, memory persistence, and fresh noise.
002 Uses Experiment 7 learned agents. No consciousness inference. Python stdlib.
003 """
004 import sys, math, random, csv, statistics
005 # Local import: keep this script beside voha_experiment7_identity.py
006 from voha_experiment7_identity import sim, K, cosine, probs
007 N=80; POST=320; MODES=['moderate_noise','greedy','random']
008 conditions=['same_noise','fresh_noise','fresh_noise_half_memory','fresh_noise_wiped','fresh_noise_no_learning']
009
010 def restart(v,n,seed,condition,context):
011     a=v.copy(); counts=n.copy();
012     if condition=='fresh_noise_half_memory':
013         mask=random.Random(555+seed).sample(range(K),K//2)
014         for j in mask: a[j]=0.;counts[j]=0
015     if condition=='fresh_noise_wiped':a=[0.]*K;counts=[0]*K
016     rng=random.Random(101+seed*71 if condition=='same_noise' else 900010+seed*89)
017     for t in range(POST):
018         noise=[rng.gauss(0,1) for _ in range(K)]
019         j=max(range(K),key=lambda i:a[i]+0.12/math.sqrt(counts[i]+1)+0.32*noise[i])
020         if condition !='fresh_noise_no_learning':
021             # same synthetic learning environment for paired streams, stochastic reward
022             reward=context[j]+rng.gauss(0,.16)
023             counts[j]+=1;a[j]+=(reward-a[j])/counts[j]
024     return a
025
026 rows=[]
027 for mode in MODES:
028     for seed in range(N):
029         v,n=sim(seed,mode,steps=1200)
030         # identical task environment and post-restart reward potential across conditions
031         task_rng=random.Random(27000+seed)
032         task=[task_rng.uniform(-.6,1.0) for _ in range(K)]
033         probe_rng=random.Random(45000+seed)
034         probe=[probe_rng.gauss(0,.25) for _ in range(K)]
035         baseline=probs(v,probe)
036         for condition in conditions:
037             a=restart(v,n,seed,condition,task)
038             p=probs(a,probe)
039             l1=sum(abs(x-y) for x,y in zip(p,baseline))/2
040
041         rows.append(dict(mode=mode,seed=seed,condition=condition,similarity=cosine(baseline,p),distribution_shift=l1,dominant_choice_changed=int(max(range(K),key=lambda x:p[x])!=max(range(K),key=lambda x:baseline[x]))))
042         # exact restoration consistency: same input state, no fresh learning, same probe -> exactly same choice probabilities
043         out='voha_experiment8_restart_results.csv'
044         with open(out,'w',newline='') as f:
045             w=csv.DictWriter(f,fieldnames=rows[0]);w.writeheader();w.writerows(rows)
046         for mode in MODES:
047             print('\n',mode)
048             for c in conditions:
049                 sub=[r for r in rows if r['mode']==mode and r['condition']==c]
050                 print(c,'cosine',round(statistics.mean(r['similarity'] for r in sub),4),'shift',round(statistics.mean(r['distribution_shift'] for r in sub),4),'changed',round(statistics.mean(r['dominant_choice_changed'] for r in sub),3))
051         print('rows',len(rows),'file',out)
```

CSV output catalogue

Complete output data remain in the ZIP. The samples below show the exact columns and first two records, for inspection without loading the files. Large decision logs are deliberately not reproduced in full.

voha_experiment4_decisions.csv

File size: 17,898,182 bytes

Columns: seed, mode, step, task, offered_expression, training_before, offered_training, accepted, training_after, audit_holdout_after, offered_nodes

Sample row 1: 0, UAI, 0, xor_ab, ('or', ('not', ('v', 0))), ('v', 2)), 0.6364, 0.4545, 0, 0.6364, 0.2, 4

Sample row 2: 0, UAI, 1, majority_abc, ('and', ('or', ('not', ('v', 0))), ('v', 2))), ('v', 2)), 0.7273, 0.7273, 0, 0.7273, 0.8, 6

voha_experiment4_results.csv

File size: 10,981 bytes

Columns: seed, mode, final_train_accuracy, final_holdout_accuracy, final_transfer_successes, ever_intermediate_successes, accepted, rejected, attempts, initial_atom_train_accuracy, mean_final_nodes

Sample row 1: 0, UAI, 0.7272727272727273, 0.5, 0, 0, 9, 691, 700, 0.6818181818181819, 7.5

Sample row 2: 0, ConAI, 0.5227272727272727, 0.45, 0, 0, 700, 0, 700, 0.6818181818181819, 12.75

voha_experiment5_decisions.csv

File size: 2,322 bytes

Columns: mode, offer_number, task, observations, time, size, accepted, resident_units

Sample row 1: UAI, 1, 27, 0, 4, 4, 0, 0

Sample row 2: UAI, 2, 5, 0, 8, 11, 0, 0

voha_experiment5_results.csv

File size: 49,285 bytes

Columns: seed, mode, regime, successful_tasks, local_hits, cold_fetches, generic_fallbacks, local_hit_rate, resident_units, resident_programs, archive_programs, accepted, rejected, evictions, modeled_compute, modeled_lookup, modeled_training, modeled_total_cost, modeled_cost_per_task, modeled_speed_tasks_per_1000

Sample row 1: 0, UAI, familiar, 1200, 858, 0, 342, 0.715, 77, 7, 7, 293, 0, 2568, 252.0, 138.2, 2958.2, 2.4651666666666667, 405.6520857278075

Sample row 2: 0, UAI, shifted, 1200, 46, 0, 1154, 0.03833333333333333, 77, 7, 7, 293, 0, 5816, 252.0, 138.2, 6206.2, 5.171833333333333, 193.3550320647095

voha_experiment6_decisions.csv

File size: 14,548 bytes

Columns: mode, step, phase, option, is_composite, noise, reward, cumulative, distinct_compositions

Sample row 1: greedy, 0, 0, 0, 0, -0.0, 0.4225, 0.4225, 0

Sample row 2: greedy, 1, 0, 0, 0, 0.0, 0.3629, 0.7855, 0

voha_experiment6_results.csv

File size: 66,777 bytes

Columns: seed, mode, cumulative_reward, mean_reward, distinct_useful_compositions, reused_useful_compositions, distinct_choices, final_choice_accuracy, post_shift_mean_reward, preference_concentration, unique_operators_created

Sample row 1: 0, greedy, 758.6138498806328, 0.42145213882257376, 0, 0, 1, 0.24422504454047375, 0.2942310274316755, 1.0, 0

Sample row 2: 0, low_noise, 1328.9225555809667, 0.7382903086560926, 1, 1, 1, 0.7309838583526136, 0.8330367508589624, 1.0, 0

voha_experiment7_results.csv

File size: 1,057 bytes

Columns: mode, n, steps, recognition_accuracy, same_identity_similarity, different_identity_similarity, recognition_margin, half_memory_similarity, full_memory_wipe_similarity, half_memory_recognition, full_wipe_recognition, distinct_learned_actions

Sample row 1: greedy, 100, 1600, 0.02, 0.43418006741936405, 0.42654670233440384, 0.007633365084960209, 0.45548686226612184, 0.45486667211200693, 0.02, 0.01, 1.06

Sample row 2: low_noise, 100, 1600, 0.14, 0.5671378932352085, 0.3308939376038098, 0.23624395563139866, 0.5412683573274604, 0.41259950797124706, 0.1, 0.01, 2.6

voha_experiment8_restart_results.csv

File size: 78,796 bytes

Columns: mode, seed, condition, similarity, distribution_shift, dominant_choice_changed

Sample row 1: moderate_noise, 0, same_noise, 0.9564674214339145, 0.1948879003625106, 0

Sample row 2: moderate_noise, 0, fresh_noise, 0.7277678916047272, 0.2838219879791667, 1

Code review checklist

- Check paired seeds, environment and reward schedules before comparing policies.
- Audit all synthetic resource costs, especially cold-fetch versus generic-fallback assumptions.
- Compare learning policies with stronger modern baselines, not only compulsory-update FIFO.
- Repeat Experiment 7 across independently generated cohorts and probe contexts; quantify uncertainty and tie handling.
- Examine the complete-erasure chance baseline as an expected consequence of the fingerprint method.
- Separate memory restoration, fresh experience, and true process restart in Experiment 8.
- Preregister Experiment 9 before implementing rhythmic stimulation and noise sweeps.
- Preserve original source and raw CSVs when modifying any script.
- Do not interpret behavioral recognition as subjective consciousness.