

Supplementary Material: OIBench 1.0

The Operator Intelligence Benchmark: Domain Construction, Constraint Contraction, and Certified Solution Emergence in Neurosymbolic AI
Thanh Trung Phan

S1. Exact task semantics

OIBench tasks are finite constraint-satisfaction instances with fully executable semantics. A task record defines a reference candidate domain X , a unique solution x^* , and a set of constraints c_i that are all satisfied by x^* . The generator keeps constraints only when they reduce the current domain without deleting x^* . If non-direct relational constraints do not isolate the solution, direct equality clues are added. Redundant valid constraints are then appended so that selection order remains nontrivial.

Permutation tasks enumerate permutations of 5-7 symbols. Boolean tasks enumerate 9-12 bits. Modular tasks enumerate three bounded integer coordinates. Coloring tasks enumerate 3-color assignments on 6-8 nodes. The exact Python predicates in `code/oibench_core.py` are normative for the release.

S2. Constraint vocabulary

Family	Constraint types	Examples
Permutation	eq, neq, lt, absdiff_eq, adj, parity, sum2_eq, absdiff_le	$x_i=v$; $x_i < x_j$; $ x_i - x_j =1$
Boolean	bit_eq, xor, imp, clause_or, count_eq, parity_subset	$b_i \text{ XOR } b_j=r$; exact subset count
Modular	sum_mod, linear_mod, sum_eq, product_mod, order, coord_neq	$a.x+b.y+c.z = r \bmod m$
Coloring	edge_neq, node_eq, color_count, node_same, all_distinct3	$c_i \neq c_j$; exact color count

S3. Generator integrity checks

Every generated instance is exhaustively re-evaluated after its constraint pool is finalized. A task is released only if the conjunction of all serialized constraints leaves exactly one state and that state matches the stored solution. The `solution_hash` field is the SHA-256 digest of the canonical JSON encoding of that solution. The package verifier additionally checks split sizes against the manifest.

S4. Metrics and reference cost model

The reference executor counts one candidate evaluation whenever a constraint predicate is tested on one currently surviving candidate. Applying operator c_t to D_t therefore costs $|D_t|$ primitive evaluations. Exhaustive certification is normalized as $|D_0|$ times the number of serialized constraints. This deliberately simple cost model is independent of Python implementation details and is appropriate for comparing contraction policies that share the same symbolic predicates.

$$\text{Cost}(\text{method}) = \sum_t |D_t|.$$

$$\text{Relative cost} = \text{Cost}(\text{method}) / (|D_0| * \text{number_of_constraints}).$$

S5. Neural selector

The neural baseline predicts the immediate gain of an operator. Its input contains four family indicators, one-hot operator type, current and initial log2 domain size, normalized step position, remaining-operator count, operator arity,

and summary statistics of serialized numeric parameters. The target is $-\log_2(|\Omega(D)|/|D|)$. A scikit-learn MLPRegressor with hidden layers (64,32), ReLU activation, early stopping, and random state 20260913 is fit on 10,208 operator-state examples generated from 180 training tasks. The model is intentionally compact. It is serialized as `results/neural\_selector.joblib`.

Development diagnostic: MAE=0.404 gain bits; R2=0.717 on 2687 operator-state examples.

S6. Statistical analysis

All reported hypothesis tests are paired by test task. One-sided Wilcoxon signed-rank tests compare candidate-evaluation costs. Bootstrap 95% intervals for median cost ratios use 5,000 task resamples with release seed 20260913.

Comparison	p-value	Median task-cost ratio	Interpretation
neural < random cost	5.150e-14	0.736	lower cost supported
probe32 < random cost	6.964e-34	0.650	lower cost supported
oracle_greedy < random cost	7.180e-35	0.640	lower cost supported
static < random cost	8.118e-01	1.019	not significant
neural > probe32 cost	1.609e-15	1.035	higher cost supported

S7. Extended family results

Method	Family	Success	Median cost ratio	Mean steps
neural	boolean	1.000	0.079	11.44
neural	coloring	1.000	0.138	13.36
neural	modular	1.000	0.080	5.98
neural	permute	1.000	0.091	6.26
oracle_greedy	boolean	1.000	0.072	7.82
oracle_greedy	coloring	1.000	0.074	6.02
oracle_greedy	modular	1.000	0.078	3.70
oracle_greedy	permute	1.000	0.089	4.30
probe32	boolean	1.000	0.073	7.88
probe32	coloring	1.000	0.078	6.02
probe32	modular	1.000	0.081	3.74
probe32	permute	1.000	0.092	4.32
random	boolean	1.000	0.117	14.72
random	coloring	1.000	0.123	11.32
random	modular	1.000	0.137	7.82
random	permute	1.000	0.131	7.58
static	boolean	1.000	0.137	14.98
static	coloring	1.000	0.124	11.24
static	modular	1.000	0.141	7.24
static	permute	1.000	0.125	8.26

S8. Benchmark-use recommendations

For publishable benchmark comparisons, systems should report certificate validity before efficiency. A method that contracts quickly by deleting the solution should be treated as failed regardless of cost. Learned policies should be

evaluated on the fixed hidden test split and should disclose whether task-family labels or operator-type identities are used. Track B systems should provide either an extensional candidate set or an executable membership predicate. Track C systems should provide an executable derived predicate and a proof or empirical exhaustive check demonstrating soundness on the finite reference domain.

Leaderboard deployment should remove `solution` from the public test file and evaluate server-side. The research package retains solutions because the present goal is transparent method development and reproducibility.

S9. Failure modes

OIBench distinguishes several failure modes: invalid contraction (solution deleted), incomplete contraction (more than one state remains), certificate mismatch, excessive symbolic work, over-specialization to one reasoning family, and poor transfer to unseen operator types. The coloring result of the compact neural baseline illustrates the last two issues: final correctness remains perfect because the verifier is exact, while search efficiency degrades relative to random ordering.

S10. Reproduction commands

Install dependencies with ``python -m pip install -r requirements.txt``, run ``python code/smoke_test.py``, regenerate the data with ``python code/generate_benchmark.py --outdir data --seed 20260913 --per-family 250``, and run the baselines with ``python code/run_baselines.py --root .``. The package includes the exact released outputs so that reviewers can inspect results without retraining the neural selector.