WHAT DO HYPERGRAPH BENCHMARKS MEASURE? FEATURE PATHS, EVALUATION PROTOCOLS, AND MODULAR CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hypergraph neural networks (HNNs) are compared by node-classification accuracy on a fixed set of benchmarks, and the differences are read as evidence about their aggregation operators. We test that reading from two sides.

First, we ask what the benchmarks measure. Ablating the released ED-HNN code at its published configurations, and a factorial over datasets, hyperedge constructions and architectures, show that accuracy depends mainly on whether a node's own features reach its output — often through a single preprocessing line — while the aggregation operator moves it by under two points.

Second, we ask what the benchmarks cannot measure. We build HYPERCSP, constraint hypergraphs with known answers, a majority-class floor and a non-neural reference, evaluated so that input labels cannot be copied and no test label touches model selection. On modular constraints, additive aggregators with linear messages solve none of 720 runs, while an exclusive-product aggregator (PRODSET) solves 149 of 240. Controls locate the difference in a receiver-specific interaction among the other members: the same product computed from additive power sums performs like PRODSET, averaging inside the same pipeline fails, a receiver-exclusive Deep Sets learns the binary constraint but rarely the ternary one, and every additive arm learns a cardinality constraint. A naive evaluation protocol hides this picture, overstating PRODSET and erasing the cardinality result.

Carried back to seven benchmarks, PRODSET is never better than mean aggregation, even under a protocol in which it recovers parity constraints injected into a real hypergraph; on LDPC decoding, where such constraints exist, it recovers much of belief propagation's gain and additive arms none. Under the probes and protocols we test, the capability that separates these operators is one the benchmarks do not exercise. We release HYPERCSP, the protocol and all code.

1 INTRODUCTION

A hypergraph neural network is usually specified by how it aggregates the members of a hyperedge. AllSet (Chien et al., 2022) casts the operator as a pair of learned set functions; ED-HNN (Wang et al., 2023) derives one from equivariant diffusion; HGNN (Feng et al., 2019) and UniGCNII (Huang & Yang, 2021), the latter carrying GCNII's initial residual (Chen et al., 2020), vary the normalisation and the residual. Papers are compared by node-classification accuracy on a fixed collection of benchmarks, reported differences are typically one to three points, and the comparison is read as evidence about the operators. We argue that the reading is not supported, from two directions that meet.

What the benchmarks measure. Each HNN we examine also contains a route by which a node's own features reach its output without being averaged into its hyperedges: an additive residual, an initial-residual (restart) blend, a concatenation, or — in the case we find most striking — a preprocessing step that adds the singleton hyperedge $\{v\}$. In the configurations we examine, which of these supplies the route matters little; whether *any* does is decisive (§3), and the aggregation operator moves accuracy by under two points on average.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

What they cannot measure. The operators these comparisons rank (HGNN, AllSet, ED-HNN, UniGCNII) share an algebraic form: the message a hyperedge sends is a sum — possibly weighted, possibly passed through learned maps — over its members. Tensor-based HNNs model joint interactions of hyperedge members (Wang et al., 2025), and factor-graph networks can represent belief propagation (Zhang et al., 2020); neither is what the standard benchmark tables compare. Belief propagation on a parity constraint is multiplicative: the log-likelihood-ratio message to one member satisfies $\tanh(\mu_{\text{out}}/2) = \prod_i \tanh(\mu_i/2)$, the product running over the other members' incoming messages μ_i. Sum-decomposition is universal over multisets (Zaheer et al., 2017), so no representational impossibility follows and we claim none; the question is what gradient descent learns from partial labels, and we answer it empirically on HYPERCSP, a suite where the answer is known by construction (§4–6). The answer is specific rather than sweeping: additive aggregation learns a cardinality constraint readily, and fails on the *periodic* (modular) constraints that a product aggregator learns. Whether this matters in practice is a question the benchmarks cannot answer; §7 carries the product aggregator back to them, measures the power of that probe, and shows the separation reappear on a real decoding task.

Contributions.

- **An attribution on the original authors' code** (§3): on the four shallow published ED-HNN configurations one preprocessing line is worth up to 30.5 pp and every architectural feature path at most 0.15 pp; on the deep one the restart is worth 49 pp.
- **HYPERCSP** (§4), generated constraint hypergraphs — parity, modular sums, cardinality — with uninformative features, a majority-class floor and a non-neural reference for every task, and an evaluation protocol in which the objective cannot be met by copying input labels and nothing about test nodes enters training or checkpoint selection.
- **A specific separation** (§5–6): on modular constraints three linear-message additive aggregators solve 0 of 720 runs and an exclusive-product aggregator (PRODSET) 149 of 240; exact and matched controls attribute the gap to a receiver-specific interaction of the other members, which the product builds in and a receiver-exclusive Deep Sets learns only for the binary constraint; on cardinality every arm succeeds.
- **A null transfer result with measured power** (§7): PRODSET is never better on seven benchmarks under either protocol, although it recovers parity constraints injected into a real hypergraph; on LDPC decoding it recovers up to 90% of belief propagation's error reduction where additive arms recover none.
- **A measurement finding** (§8): a naive protocol inflates the product's solve rate from $164/240$ to $228/240$ and hides the cardinality result, and identical commands do not reproduce individual runs.

We claim neither residual connections nor multiplicative aggregation as novel; the contribution is the attribution and the diagnosis.

2 RELATED WORK

Hypergraph neural networks and benchmark critiques. The dominant framing is aggregation design (Feng et al., 2019; Chien et al., 2022; Wang et al., 2023). That standard benchmarks fail to separate architectures is established for graphs: reported GNN rankings do not survive controlled re-evaluation (Shchur et al., 2018), and much reported progress in graph classification disappears under matched protocols (Errica et al., 2020). For hypergraphs, Li et al. (2025) show that current HNNs struggle on heterophilic hypergraphs and release the benchmarks we use, Telyatnikov et al. (2025) question whether datasets lifted from graphs exercise hypergraph-specific structure, and DHG-Bench (Li et al., 2026) evaluates 17 HNNs on 22 datasets under one protocol. We do not add scale; we add interventions and tasks with known answers: §3 identifies *which* component the accuracy depends on (the node-feature path, often supplied by preprocessing), and §7 tests whether an operator with a capability the others lack can find anything to use.

Set-function expressivity. Sum-decomposition is universal over multisets (Zaheer et al., 2017), the sum retains what mean and max discard (Xu et al., 2019), and universality depends on the latent

Table 1: Released ED-HNN code at its *published* per-dataset configuration (plotted in Figure 3). FULL reproduces the paper's own accuracy within 1.5 pp, which is what licenses the rest of each row. On the four shallow configurations, removing one preprocessing line costs up to 30.5 pp and removing every architectural feature path at most 0.15 pp, because they contain none. Pubmed, the only deep configuration, inverts the pattern: its restart is worth 49 pp.

dataset	published	FULL	−selfloop	−skip	−path
Citeseer (L=1, α=0)	73.70	73.12	**42.61**	72.97	73.12
Cora-CC (L=1, α=0)	80.31	81.27	**59.85**	81.27	81.27
Cora-CA (L=1, α=0)	83.97	85.44	**76.75**	85.55	85.55
DBLP (L=1, α=0)	91.90	91.89	91.07	91.93	91.87
Pubmed (L=8, α=0.5)	89.03	88.92	88.83	**39.89**	**39.89**

dimension (Wagstaff et al., 2019). We therefore claim a measured difference in what models learn from partial labels, not in what they can represent.

Learned message passing on constraint graphs. The closest prior work is outside the hypergraph literature. Neural belief-propagation decoders (Nachmani et al., 2016; 2018) attach learned weights to sum-product messages on a Tanner graph — a hypergraph of parity constraints — and factor graph neural networks (Zhang et al., 2020) generalise this to learned factor-graph inference; both are multiplicative in our sense and are not among the operators we call "the HNN literature". We do *not* claim PRODSET as novel: it is close to a neural BP decoder written as a hypergraph layer. The contribution is the surrounding diagnosis (§3, §5, §7), not the aggregator.

3 WHAT PUBLISHED BENCHMARKS MEASURE: FEATURE PATHS

We call a *node-feature path* any route by which node v's representation contributes to its own update other than through the aggregate of the hyperedges containing v. In the implementations we study these take four forms: an additive residual; a restart blend $(1-\alpha)f(x)+\alpha x_0$; a concatenation of x_v inside the hyperedge-to-node update; and a *data-level* path, where preprocessing adds the singleton hyperedge $\{v\}$.

On the authors' own code and configurations. We run the released ED-HNN implementation, unmodified except for an ablation switch, at its published per-dataset hyperparameters. Reproducing the published accuracy is a precondition — an ablation measured against a baseline that is not training says nothing — and Table 1 reports both. Parameter shapes are never changed; we zero the relevant input rather than delete a weight, so capacity is fixed.

The architectural ablations are exact no-ops, and that is the finding. For the four shallow datasets the published commands set `--restart_alpha 0.0` and `--MLP2_num_layers 0`, and with the latter the released code sets $W_2 = \lambda X.\, X[\ldots, d_{\text{in}}{:}]$, discarding the node term from the concatenation. These configurations have no architectural feature path at all: beyond hyperedge averaging, their entire route from x_v to $\hat{y}_v$ is the preprocessing self-loop.

The deep configuration inverts it, for a reason that is not the obvious one. On Pubmed (L=8, α=0.5) deleting the self-loop costs 0.09 pp while deleting the restart costs 49.0 pp. The obvious reading — that message-passing depth makes the restart necessary — is not what the data show (Appendix I). Holding propagation depth fixed at L=8 and changing only the per-layer MLP depth from $0/0/1$ to $2/2/2$ moves the restart's value from $+0.67$ to -49.2 pp on Citeseer and from -0.09 to -43.8 on Cora-CC: what makes it necessary is the depth of the *parameterised* stack, not the number of propagation steps.

So the two paths do different jobs and neither is about aggregation: the self-loop supplies a node's own features to an aggregation that would otherwise average them away; the restart enables optimisation of a deep stack, the ordinary role of residuals (He et al., 2016).

Table 2: Hypergraph parity with features that noisily encode the latent bit. Left: simplified re-implementations of published architectures (the scatter-mean versions of Appendix D, not the released code; 5 seeds, AUC), ordered by the directness of their feature path. Right: ablating one model over {convolution, ordered scan, restart} × {residual on, off}, 40 runs, reported as in Table 3 (solve = AUC ≥ 95). With a residual present all six aggregation combinations solve every run; without one the aggregation-only variants solve none and the restart-only variant $33/40$.

published architecture	AUC	our ablation (40 runs)	AUC	solved
MLP (no structure)	99.94 ± 0.09	residual on, all 6 combos	100.00 ± 0.00	$\mathbf{40/40}$ each
AllSet	$\mathbf{50.37 \pm 1.13}$	residual off, restart on	91.25 ± 19.01	$33/40$
ED-HNN	76.84 ± 17.68	residual off, convolution only	49.83 ± 1.45	$0/40$
UniGCNII	72.07 ± 17.86	residual off, sequence only	51.19 ± 7.13	$0/40$

A controlled task isolates the mechanism. We also use a task where beyond-pairwise information provably exists and pairwise information provably does not: each node carries a latent bit g_v, and a triple is a valid hyperedge iff $g_u \oplus g_v \oplus g_w = 0$, so for any pair exactly half of all third nodes complete a valid triple whatever their bits. Two non-neural controls confirm it (a pairwise co-occurrence scorer at 50.4 AUC, an exact triple lookup at 50.0); earlier versions were discarded when these controls revealed a negative sampler that inverted the signal and a pairwise shortcut that let one architecture reach a meaningless 100.0.

The right-hand columns of Table 2 are the attribution: six combinations of convolution, ordered scan and restart solve all 40 runs each with an additive residual; without one the two aggregation-only variants solve none and the restart-only variant 33 (Figure 2). The hierarchy also predicts the (re-implemented) published architectures: our no-path variant (49.8) matches AllSet (50.4), and our restart-only variant, bimodal at $33/40$, sits in the same unstable regime as ED-HNN (76.8 ± 17.7) and UniGCNII (72.1 ± 17.9).

And now remove the features. If the same latent bits must be recovered from the incidence alone — features pure noise — *no* method exceeds chance: MLP 50.61, AllSet 49.90, ED-HNN 50.75, UniGCNII 50.34, ordered scan 50.99. Yet the information is present and efficiently extractable: each hyperedge is a $\mathrm{GF}(2)$ equation and Gaussian elimination attains 100.00 ± 0.00. Nor is the gap about locality: sum-product belief propagation is strictly local and also reaches 100.0 at density $m/n \geq 2$, tracking the fraction of nodes the constraints determine below it (Appendix A).

A 50-point gap separates the entire architecture class from two algorithms that solve the task, one of them local; the remaining difference is the *form of the message*, which the rest of the paper makes its object.

How much is left for the operator. A balanced factorial study over 7 datasets, 6 hyperedge constructions and 5 architectures (Appendix B, with the full per-dataset variance decomposition) attributes a mean absolute spread of 1.79 pp to the architecture and 1.70 pp to the construction, while seed noise accounts for 40% of the variance averaged over datasets (a mean of per-dataset shares, not a pooled decomposition) — and for 87% on Pokec and 68% on Twitch-gamers, where neither factor is resolvable at all. Ablating every feature path across four architectures and six datasets (Appendix H) costs 6.4–32.1 pp wherever models learn, and leaves the four architectures within 0.2–4.6 pp of one another on any dataset. The architecture is not always negligible — it carries 84% of the variance on Actor and interacts with the construction for 49% on Walmart-Trips (Appendix B) — but on most datasets published HNN comparisons are, to a first approximation, comparisons of feature-path strength. A large ablation loss shows that a path is *necessary*, not what share of accuracy it contributes when components interact, and our factorial does not cross feature paths with architectures.[1]

[1] We first ran this ablation at non-published settings (L=2, width 64, α=0.5), where FULL scored 22.5–29.3 against published values of 73.7 and 80.3 — the model was not training. Ablations against a broken baseline are uninformative, so we exclude those runs rather than report them.

4 HYPERCSP: CONSTRAINT TASKS WITH FLOORS AND REFERENCES

Each task samples a ground-truth labelling g over n nodes, then samples r-subsets and *keeps only those that g satisfies*, yielding m hyperedges. A fraction ρ of nodes is revealed; the model predicts the rest. Node features are a noise channel, a revealed-flag, and the revealed label one-hot — so all information about a hidden node lies in the incidence structure. We use three families:

family	constraint	non-neural reference
`parity`	$\sum_{v\in e} g_v \equiv 0 \pmod 2$	$\mathrm{GF}(2)$ elimination; belief propagation
`mod3`	$\sum_{v\in e} g_v \equiv 0 \pmod 3$	$\mathrm{GF}(3)$ elimination
`onehot`	exactly one $v \in e$ has $g_v = 1$	unit propagation

`mod3` is included deliberately: product-of-$\tanh$ is parity's belief-propagation rule, whereas $\bmod 3$ needs a Fourier/circular convolution, so a method that wins on `mod3` *without being re-derived* exhibits a property of its aggregation family, not alignment to one problem. The hyperedges matter as such: two hypergraphs with identical pairwise co-occurrence can have different constrained answers (Appendix A).

Floors and references. Reporting accuracy alone is not interpretable here: on `onehot` the labels are imbalanced, so 67% is available by ignoring the hypergraph entirely. Every cell therefore carries the majority-class accuracy on the test nodes (floor; computed from test labels, so a description of class balance rather than a deployable predictor, which a predictor trained on anchors alone would approach but need not reach) and the fraction of hidden nodes an exact solver determines and gets right (reference). The reference is not an accuracy upper bound: undetermined nodes guessed at chance add $(1-q)/p$ for a $\mathbb{Z}_p$ task with determined fraction q, which matters only in the diluted regime of Appendix C.

Evaluation protocol. Every revealed node is an anchor: at evaluation its label is an input feature. During training a random 30% of anchors per epoch have that input withheld, and the loss is computed on exactly those nodes, so the objective cannot be met by copying an input label and must be met by propagating from the remaining anchors. Hidden nodes are split into a validation part (25%), which selects the checkpoint, and a test part, which is the only quantity reported; nothing about test nodes touches training or selection. §8 shows what a naive alternative changes.

Regimes. At density $m/n = 3$ the constraint system determines the labelling almost surely, and both $\mathrm{GF}(2)$ elimination and belief propagation reach 100.0; at $m/n \le 1$ neither does, tracking the fraction of determined nodes. We report the identified regime ($m/n = 3$, $n = 300$, $r = 3$) at anchor fractions $\rho \in \{0.25, 0.40, 0.60\}$; Appendix G gives the full supervision curve.

5 WHICH CONSTRAINTS ADDITIVE AGGREGATION LEARNS

We hold everything fixed except the aggregator: the same feature path ($h \leftarrow h + \mathrm{relu}(\mathrm{agg}(h))$), depth, width, optimiser, schedule and protocol. Four additive arms span the family: the mean; the sum, which keeps the count the mean discards (Xu et al., 2019); an *exclusive* sum $\sum_{v\neq u} W h_v$ with PRODSET's receiver exclusion, in/out maps and near-identity initialisation; and a full Deep Sets $\rho(\sum_v \phi(h_v))$ with MLP encoders. Table 3 reports the solve rate — the fraction of 40 runs whose validation-selected checkpoint reaches $\ge 95\%$ on held-out test nodes — because the outcome is bimodal (§8).

The modular families separate; the cardinality family does not. On `parity` and `mod3` the mean, sum and exclusive sum never reach the criterion (no run above 83%) and their average stays within 6.2 points of the majority-class floor, below it in 16 of 18 arm-cells, while PRODSET solves 149 of 240 runs; against each linear arm every cell separates at $p \le 8 \times 10^{-6}$ (paired exact McNemar). The exclusive-sum arm is the control the comparison needs: it has PRODSET's receiver-dependent messages, its maps and its initialisation, and solves nothing. Three further controls locate the difference (Appendix G). Keeping PRODSET's entire pipeline — $\tanh$ input with temperature,

Table 3: HYPERCSP, $n = 300$, $r = 3$, $m/n = 3$, 8 layers, corrected protocol (§4), 40 runs per cell; entries are solve rates. All arms share instances and seeds, so comparisons with PRODSET are paired (exact McNemar). On the modular families the three linear-message additive arms solve 0 of 720 runs; the matched exclusive mean, which keeps PRODSET's entire message pipeline and replaces only the product, solves 6 of 240; Deep Sets reaches the criterion only at the densest supervision. The matched control was run on the modular families only (—). On the cardinality family every arm succeeds. The floor is the majority class of the test nodes the networks are scored on; the reference is the fraction of hidden nodes elimination determines. Per-run accuracies: `nextdir_phase3_table3_masked_raw.json`, `..._rho060_masked_raw.json`.

				linear additive			matched		
family	ρ	floor	reference	mean	sum	excl. sum	excl. mean	Deep Sets	PRODSET
parity	0.25	53.71	100.00	0/40	0/40	0/40	0/40	0/40	**23/40**
	0.40	53.65	100.00	0/40	0/40	0/40	0/40	0/40	**24/40**
	0.60	53.56	100.00	0/40	0/40	0/40	6/40	21/40	30/40
mod3	0.25	37.03	99.98	0/40	0/40	0/40	0/40	0/40	**18/40**
	0.40	37.54	99.97	0/40	0/40	0/40	0/40	0/40	**23/40**
	0.60	38.89	99.96	0/40	0/40	0/40	0/40	4/40	**31/40**
modular, pooled					**0/720**		6/240	25/240	**149/240**
onehot	0.25	66.97	99.99	36/40	18/40	22/40	—	36/40	37/40
	0.40	66.52	99.99	33/40	28/40	34/40	—	37/40	39/40
	0.60	67.33	100.00	32/40	36/40	32/40	—	36/40	37/40

clamp, $2\,\mathrm{artanh}$ output, maps, initialisation — but averaging the other members instead of multiplying them solves 6 of 240 modular runs. The same pipeline computing the *same* exclusive product from additive power sums (Newton's identities; identical to PRODSET in exact arithmetic, checked to 10^{-14} in float64) solves 142 of 240, no cell different from PRODSET ($p \geq 0.24$): how the product is computed does not matter, which function the message computes does. And Deep Sets made receiver-exclusive, $\rho(\sum_{v \neq u} \phi(h_v))$, solves 62 of 240 against 25: it matches PRODSET on `parity` at $\rho \geq 0.40$ (21 and 27 of 40, $p \geq 0.63$) but not on `mod3` or at $\rho = 0.25$ ($p \leq 5 \times 10^{-4}$). The separation is therefore not between writing a message as a sum or a product, but between messages that form a receiver-specific nonlinear interaction of the other members and messages that do not; the product builds that interaction in, a learned encoder must find it, and does so for the binary constraint with enough anchors. On `onehot`, by contrast, every additive arm reaches the criterion in 18 to 36 of 40 runs in every cell. An intuition we do not prove: the onehot rule is a threshold on the count of the other members, which one linear aggregate and a ReLU represent, whereas the modular rules are periodic in that count.

Plain Deep Sets. Sum-decomposition represents any multiset function (Zaheer et al., 2017); without receiver exclusion Deep Sets reaches the criterion in no run at $\rho \leq 0.40$, although its `parity` accuracy rises from 51.6 to 63.5, and at $\rho = 0.60$ solves 21/40 on `parity` ($p = 0.09$ against PRODSET) but 4/40 on `mod3`. These are learning curves under one training budget, not sample-complexity bounds (Appendix G gives the full curve and the 90/95/99% criteria).

Is this parity being hard? Parity is the canonical representable-but-hard-to-learn function for gradient methods (Shalev-Shwartz et al., 2017). Three facts place the obstruction in the architecture: the same optimiser on the same instances reaches the reference with PRODSET (and, at $\rho = 0.60$, Deep Sets); `mod3`, whose rule product-of-$\tanh$ does not implement, separates at least as strongly; and the matched additive controls rule out receiver exclusion and initialisation. We make no hardness claim for this planted, partially labelled task.

6 PRODSET

PRODSET replaces the additive aggregate by an exclusive product over hyperedge members. Writing $t_v = \tanh(W_{\text{in}} h_v / 2\tau)$ with a learned temperature $\tau > 0$, the message returned to member u of

hyperedge e is

$$m_{e\to u} \;=\; W_{\text{out}} \cdot 2\,\operatorname{artanh}\Big(\textstyle\prod_{v\in e,\, v\neq u} t_v\Big),$$

with $\tanh$, artanh and the product taken elementwise over channels, computed exactly by prefix/suffix products along the arity axis; node u adds the sum of its incoming messages to its state through the residual update of §5. For real hypergraphs, whose hyperedges vary in size, we group hyperedges by arity and apply the same exact computation per group, with no padding and no member subsampling.

Which ingredients matter. Table 8 (Appendix G) isolates them under the corrected protocol. The plain exclusive product solves $36/40$ where the mean solves none; a learned temperature ($38/40$) and near-identity initialisation ($37/40$) change nothing measurable (Fisher $p \geq 0.68$; not equivalence results). Depth matters: the same model solves $37/40$ at $L = 4$ and $24/40$ at $L = 8$ ($p = 0.001$), and at $L = 4$ `mod3` gives $38/40$ while every additive arm stays at 0 or 1 of 40 (Appendix G). Table 3 keeps the $L = 8$ fixed before these runs, which understates PRODSET.

The tied column shows what iteration needs. One shared rule applied 16 times — what belief propagation does — solves $9/40$ as a plain residual stack and $40/40$ once messages are damped and layer-normalised ($37/40$ with signed messages; $p < 10^{-12}$), matching the untied $L = 4$ model with one shared aggregator instead of four. Without stabilisation, weight sharing is easily mistaken for evidence against the approach (Appendix K records more variants under the naive protocol).

7 REAL BENCHMARKS: A NULL, WITH A POSITIVE CONTROL

We transfer PRODSET unchanged to standard benchmarks, again altering only the aggregator. Because a null result is only informative if the implementation works, we run a positive control through the *identical* code path and training harness: a constraint hypergraph with mixed arities $\{3, 4, 5\}$, which exercises the variable-arity grouping that the uniform tasks never touch. It separates by 44.8 pp (Table 5); the benchmarks do not. A null must be tested as one, so Table 4 also reports two one-sided tests (TOST) at the architecture effect size we measured ourselves in §3, ± 1.79 pp, and at a stricter ± 1.5 pp. The ± 1.79 margin is exploratory: it is the average operator effect in our own factorial, chosen after the fact rather than pre-registered, and a smaller gap can still reorder two particular models. Across margins of ± 0.5, 1.0, 1.5 and 1.79 pp, equivalence holds on 2, 5, 6 and 7 of 7 datasets here and on 2, 4, 6 and 6 under the anchored protocol, so the equivalence claim is about differences of a point or more, not about exact equality. Equivalence is *established* on 7 of 7 datasets at the measured margin and 6 of 7 at the stricter one; the exception is Cora-CA, where PRODSET is genuinely, if slightly, worse (-1.08 pp, 95% CI $[-1.88, -0.28]$). Under Holm correction across the seven datasets only Actor remains significant, and its sign is also against PRODSET.

How much structure could the benchmarks hide, and could this protocol see it? A null is only as strong as the detector, so we measured its power twice (Appendix C). On synthetic `parity` diluted with random hyperedges, PRODSET separates from the mean at 50% constraint hyperedges ($+8.50$ pp) and not at 25% — an observed range, not an estimated power curve — with no significant offset at zero structure ($+0.54$, $p = 0.42$). On a *real* hypergraph (Twitch-gamers, real features and hyperedges retained) with parity constraints on the real labels injected at per-node density m/n, it approaches the oracle (99.38 vs. 99.80 at $m/n = 2$) and separates down to $m/n = 0.25$ — but *only when training labels are supplied as input*. Under the standard protocol, with labels entering through the loss alone, it exploits none of the structure (50.83 at an oracle of 99.80): this propagator needs anchors, and the standard protocol gives it none. So Table 4 (standard protocol) has essentially no power to detect this kind of structure with this probe, and Table 5 repeats the seven benchmarks under the anchored protocol that does. PRODSET is never better there either: equivalence at ± 1.79 pp holds on 6 of 7 datasets, and the one difference that survives Holm correction (Cora-CA, -1.36) is against it. A probe that resolves injected parity structure at $m/n = 0.25$ on a real hypergraph thus finds none in these benchmarks — a bound on one kind of structure at comparable density, not on higher-order information in general.

[2]The MLP column is identical for Cora-CC and Cora-CA by construction: the two datasets share node features and labels and differ only in how hyperedges are built, which a structure-free model never sees.

Table 4: Real benchmarks under the standard protocol (labels enter only through the loss), 10 seeds, only the aggregator differs. Difference is PRODSET−mean over paired seeds; TOST at ± 1.79 pp (the §3 architecture effect) and ± 1.5; Holm across the seven datasets. Equivalence holds on $7/7$ and $6/7$; nowhere does PRODSET win.

	MLP	mean agg.	PRODSET	diff.	95% CI	$\text{TOST}_{1.79}$	$\text{TOST}_{1.5}$	p_{Holm}
Citeseer	71.96 ± 2.07	72.66 ± 1.47	72.09 ± 1.70	-0.57	$[-1.61, +0.47]$	0.013	0.037	1.000
Cora-CC[2]	74.92 ± 1.66	78.86 ± 1.10	78.92 ± 0.97	$+0.06$	$[-0.69, +0.80]$	< 0.001	0.001	1.000
Cora-CA	74.92 ± 1.66	83.59 ± 1.09	82.51 ± 0.94	-1.08	$[-1.88, -0.28]$	0.037	0.131	0.082
Pubmed	$\mathbf{88.32 \pm 0.54}$	88.19 ± 0.53	88.10 ± 0.64	-0.09	$[-0.33, +0.15]$	< 0.001	< 0.001	1.000
Actor	$\mathbf{83.98 \pm 0.35}$	82.50 ± 0.56	82.04 ± 0.62	-0.46	$[-0.60, -0.32]$	< 0.001	< 0.001	$< \mathbf{0.001}$
Amazon-ratings	27.66 ± 0.39	29.24 ± 0.29	29.08 ± 0.41	-0.16	$[-0.48, +0.17]$	< 0.001	< 0.001	1.000
Twitch-gamers	50.81 ± 0.65	51.15 ± 0.82	51.49 ± 1.43	$+0.34$	$[-0.33, +1.01]$	< 0.001	0.002	1.000

Table 5: The same benchmarks under the *anchored* protocol of §4, with the same code as HYPERCSP and the injection study: training labels are input features, a fresh random 30% of them is withheld and scored each epoch, and the checkpoint is selected on validation. Under this protocol PRODSET recovers injected constraint structure on a real hypergraph (Appendix C) and the positive control, run through the identical code, separates by 44.8 pp. On the benchmarks PRODSET is never better; equivalence holds on 6 of 7 at either margin, and the one difference surviving Holm correction (Cora-CA) is against it. The MLP also receives the label features.

	MLP	mean agg.	PRODSET	diff.	95% CI	$\text{TOST}_{1.79}$	$\text{TOST}_{1.5}$	p_{Holm}
positive control	53.89 ± 3.11	53.11 ± 3.71	$\mathbf{97.89 \pm 1.53}$	$+44.8$	—	—	—	—
Citeseer	70.69 ± 1.18	72.49 ± 1.62	72.08 ± 1.79	-0.41	$[-1.45, +0.63]$	0.008	0.021	1.000
Cora-CC	74.06 ± 1.45	79.50 ± 1.48	78.91 ± 1.38	-0.59	$[-1.31, +0.13]$	0.002	0.010	0.460
Cora-CA	74.06 ± 1.45	83.69 ± 0.84	82.33 ± 1.11	-1.36	$[-2.20, -0.52]$	0.139	0.357	**0.038**
Pubmed	$\mathbf{88.02 \pm 0.40}$	87.84 ± 0.47	87.95 ± 0.55	$+0.10$	$[-0.20, +0.41]$	< 0.001	< 0.001	1.000
Actor	$\mathbf{83.77 \pm 0.59}$	82.26 ± 0.58	81.94 ± 0.56	-0.31	$[-0.60, -0.03]$	< 0.001	< 0.001	0.216
Amazon-ratings	27.87 ± 0.34	29.23 ± 0.45	28.99 ± 0.36	-0.24	$[-0.52, +0.05]$	< 0.001	< 0.001	0.460
Twitch-gamers	50.79 ± 0.68	51.17 ± 0.82	51.19 ± 1.31	$+0.03$	$[-0.70, +0.75]$	< 0.001	0.001	1.000

Further readings, cost, and a caveat. Every real benchmark has 0.09–0.63 hyperedges per node (Appendix J), below the $m/n \approx 1$ at which an exact solver determines most labels on HYPERCSP — suggestive only, since a real hyperedge need not act as one independent constraint — and on Pubmed and Actor the structure-free MLP is the best arm. Against simplified scatter-mean re-implementations of AllSet, ED-HNN and UniGCNII — not the released models — each tuned per arm (Appendix D), the operators lie within 1.4–2.8 pp and PRODSET is never best. PRODSET costs 10–177× the training time of mean aggregation (Appendix E), and on continuous features its exclusive product is $\sim 10^{-4}$ at arity 3 and 10^{-6}–10^{-8} at arity 4–5 at initialisation (Appendix N) — far above float32's smallest normal number, so not an underflow, but small against the residual path; the null is therefore a null for PRODSET as implemented, not for a rescaled multiplicative operator.

Where the structure does exist, the separation appears. Constraint structure does occur in practice: a Tanner graph is a hypergraph whose hyperedges are parity checks, and every bit carries a noisy anchor. On a $(3, 6)$-regular LDPC code ($n = 300$, rate $1/2$, binary symmetric channel; Appendix M) the three additive arms are indistinguishable from the structure-free MLP and from hard decision — they use none of the structure — while PRODSET recovers 90%, 67% and 43% of belief propagation's error reduction at flip probabilities 0.03, 0.05 and 0.07 (99.69 vs. 100.00 at 0.03; every cell $p < 10^{-4}$, 10 seeds, all arms and BP scored on the same test codewords). Same aggregators and harness as Table 4, with the received word as the anchor; the difference is the data. BP stays more accurate and needs no training, so PRODSET is a diagnostic here, not a competing decoder (Appendix M gives the cost). The gap to BP widens toward the code's threshold, where BP needs its full sweep budget; unlike the identified HYPERCSP regime, where more depth did not help (§6), this is a regime where iterations are the binding resource.

How the two halves connect — a hypothesis. §3 showed that published accuracy depends on feature paths and left open why the operator matters so little. One explanation consistent with both halves: the operators differ sharply on periodic constraint structure ($0/720$ against $149/240$), and this probe, which resolves injected constraints on a real hypergraph, finds no such structure in these datasets. We have not shown that this is *the* reason; other higher-order structure may exist in the benchmarks, and a different probe might use it.

8 MEASUREMENT PROTOCOL

A copyable objective inflates one separation and hides another. Label-as-input evaluation invites two shortcuts: taking the loss on anchors whose labels are also inputs, and reporting the best test accuracy over epochs. Together they give PRODSET $228/240$ on the six $\rho \leq 0.40$ cells of Table 3 instead of $164/240$, and leave every additive arm at the floor on cardinality. Because that before–after comparison mixes both changes, we crossed them on identical runs (2×2, `parity` and `onehot`, $\rho = 0.40$, 40 runs per arm; Appendix G). The copyable loss alone produces both effects: it lifts PRODSET on `parity` from $24/40$ to $40/40$ and drops mean and Deep Sets on `onehot` from 34 and $35/40$ to $0/40$. Selecting the epoch on test labels changes little (at most $+6$ runs, only on `onehot`). Under a copyable objective an arm must propagate from initialisation, which is why near-identity initialisation looked decisive under the naive protocol ($31/40$ against $21/40$) and is immaterial under the corrected one ($37/40$ against $36/40$). All HYPERCSP and anchored real-data results use the corrected protocol; Table 2 and Appendix K do not, and say so.

Outcomes are bimodal and runs do not reproduce. Most runs either approach the rule or stay near the floor, so we report solve rates with Wilson intervals and paired exact McNemar tests, and give the $90/95/99\%$ criteria in Appendix G. Individual runs are not reproducible across processes (one command re-run three times: 84.76 ± 20.04, 93.33 ± 13.56, 99.82 ± 0.53); multi-threaded BLAS changes the reduction order, and on the CPU `OMP_NUM_THREADS=1` restores bit-exactness (Appendix L), which we do not claim for GPU runs. The same corrected-protocol cell run in two sweeps gave $24/40$ and $28/40$, so we do not interpret solve-rate differences below about 10 points.

9 DISCUSSION AND RECOMMENDATIONS

What a practitioner should do with this. On the standard benchmarks, start from a cheap aggregator with a matched feature path and adopt a more expensive operator only if it beats that baseline on validation by more than the seed noise. Where a task has check-defined structure, a multiplicative aggregator with anchors as input is a useful *diagnostic probe*; whether it is the right deployed model is a separate question, and on LDPC it is not: BP needs no training and is more accurate, though slower per codeword (Appendix M). Three reporting changes would let the evidence adjudicate aggregation design: **state the feature-path configuration, including preprocessing**; **report against a floor and a non-neural reference**; and, for label-as-input evaluation, **withhold scored labels from the input and select checkpoints on validation nodes**, since the naive alternative changed our own conclusions (§8).

10 LIMITATIONS

HYPERCSP is synthetic, with arity 3 and $n = 300$, and LDPC decoding is the one application tested. The separation is empirical and specific: additive aggregation learns cardinality, receiver-exclusive Deep Sets learns parity, and PRODSET itself solves 45–78% of modular runs at $L = 8$. Solve rates carry cross-process noise of up to $\sim$10 points. The real-data null bounds one probe and one kind of structure, not higher-order information in general; and on continuous features the operator's messages are very small at moderate arity (Appendix N), so it bounds this implementation, not the data. Three further gaps remain: apart from the released ED-HNN code of Table 1, the published operators appear only as simplified re-implementations, and none is trained on HYPERCSP; every model is trained per instance, so transfer to larger n, other arities or unseen hypergraphs is untested; and the detection-power results rest on one injected constraint family in one hypergraph.

AI USE STATEMENT

In this work we used a generative AI assistant (a large language model operated through an agentic coding tool) for research ideation and execution, for drafting sections of the paper, for polishing writing, for retrieval and discovery of related work, and for writing the code that generates the HYPERCSP synthetic datasets. We did not use generative AI to prove mathematical claims; the paper makes none. We have reviewed all AI-assisted work. Every experiment script was executed by the authors and its outputs inspected; the control experiments reported throughout (non-neural floors and references, a positive control through the identical code path, a zero-structure control, repeated sweeps of selected cells) were designed so that AI-written code could be checked against ground truth rather than trusted. They caught several defects, documented in the appendices; one they did not catch — an evaluation protocol that inflated our first results — was found in an internal review, and §8 reports what it changed. Every reference was verified by the authors against the publisher's or arXiv's own record for existence, title, and author order. We take responsibility for the final content of this work, including text, claims and artefacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only public benchmark datasets and synthetic or procedurally generated data (HYPERCSP, LDPC codewords); no human subjects, personal data or annotation were involved. The paper's findings bear on how a research community evaluates methods, and we have tried to state them with the controls and uncertainty they warrant rather than as a verdict on prior work. The compute used is reported in Appendix E; the largest single experiment is a 40-seed sweep on one GPU.

REPRODUCIBILITY STATEMENT

All results are produced by the scripts listed in Appendix O, provided as anonymised supplementary material; Appendix A specifies the HYPERCSP generation procedure and the reference solvers, Appendix F the hyperparameters of every run, and Appendix G the reproducibility properties of the measurement itself, including the environment setting that reproduces an individual CPU run bit-exactly. Real-benchmark data are the public AllSet and heterophilic-hypergraph releases. Table 1 uses the released ED-HNN code at its published configurations without modification beyond the ablation switch described in §3.

REFERENCES

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1725–1735, 2020.

Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are AllSet: A multiset function framework for hypergraph neural networks. In *International Conference on Learning Representations (ICLR)*, 2022.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations (ICLR)*, 2020.

Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3558–3565, 2019. doi: 10.1609/aaai.v33i01.33013558.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Jing Huang and Jie Yang. UniGNN: a unified framework for graph and hypergraph neural networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2563–2569, 2021. doi: 10.24963/ijcai.2021/353.

Fan Li, Xiaoyang Wang, Wenjie Zhang, Ying Zhang, and Xuemin Lin. DHG-Bench: A comprehensive benchmark for deep hypergraph learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Ming Li, Yongchun Gu, Yi Wang, Yujie Fang, Lu Bai, Xiaosheng Zhuang, and Pietro Liò. When hypergraph meets heterophily: New benchmark datasets and baseline. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 18377–18384, 2025. doi: 10.1609/aaai.v39i17.34022.

A. C. H. Ling, C. J. Colbourn, M. J. Grannell, and T. S. Griggs. Construction techniques for anti-Pasch Steiner triple systems. *Journal of the London Mathematical Society*, 61(3):641–657, 2000. doi: 10.1112/S0024610700008838.

Eliya Nachmani, Yair Be'ery, and David Burshtein. Learning to decode linear codes using deep learning. In *54th Annual Allerton Conference on Communication, Control, and Computing*, pp. 341–346, 2016. doi: 10.1109/ALLERTON.2016.7852251.

Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J. Gross, David Burshtein, and Yair Be'ery. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):119–131, 2018. doi: 10.1109/JSTSP.2017.2788405.

Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3067–3075, 2017.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. arXiv:1811.05868v2, 2018. Presented at the NeurIPS 2018 Workshop on Relational Representation Learning.

Lev Telyatnikov, Maria Sofia Bucarelli, Guillermo Bernardez, Olga Zaghen, Simone Scardapane, and Pietro Liò. Hypergraph neural networks through the lens of message passing: A common perspective to homophily and architecture design. *Transactions on Machine Learning Research*, 2025.

Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A. Osborne. On the limitations of representing functions on sets. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6487–6494. PMLR, 2019.

Fuli Wang, Karelia Pena-Pena, Wei Qian, and Gonzalo R. Arce. T-HyperGNNs: Hypergraph neural networks via tensor representations. *IEEE Transactions on Neural Networks and Learning Systems*, 36(3):5044–5058, 2025. doi: 10.1109/TNNLS.2024.3371382.

Peihao Wang, Shenghao Yang, Yunyu Liu, Zhangyang Wang, and Pan Li. Equivariant hypergraph diffusion neural operators. In *International Conference on Learning Representations (ICLR)*, 2023.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Zhen Zhang, Fan Wu, and Wee Sun Lee. Factor graph neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 8577–8587. Curran Associates, Inc., 2020.

A HYPERCSP GENERATION AND REFERENCE SOLVERS

Generation. For n nodes, arity r and target count m: draw a ground-truth labelling g; repeatedly draw a uniform r-subset and keep it iff g satisfies the family's constraint, until m hyperedges are kept or a cap of $400m$ draws is reached; then reveal a uniform random ρn nodes. For `parity` and `mod3`, g_v is uniform on $\mathbb{Z}_k$. For `onehot`, $g_v \sim \text{Bernoulli}(1/r)$, which makes the "exactly one" condition have non-vanishing acceptance probability $(\binom{r}{1}(1/r)(1-1/r)^{r-1} = 4/9$ at $r = 3)$; this choice, not the constraint itself, is what sets the majority-class floor near 67%.

Features. Each node receives $[\,\epsilon_v,\ \mathbb{1}[v \text{ revealed}],\ \text{onehot}(g_v)\cdot\mathbb{1}[v \text{ revealed}]\,]$ with $\epsilon_v \sim \mathcal{N}(0,1)$, so a hidden node's features are independent of its label and all information about it lies in the incidence structure.

Reference solvers. `parity` and `mod3` use Gaussian elimination over $\mathbb{Z}_p$: each hyperedge contributes the row $\sum_{v\in e} x_v = 0$ and each revealed node the row $x_v = g_v$; after row reduction a variable is *determined* iff its unit vector reduces to a single pivot, and the reference is the fraction of hidden nodes both determined and correct. `onehot` uses unit propagation: a positive member forces the remaining members to 0, and $r-1$ zeros force the last member to 1, iterated to fixpoint. For `parity` we additionally run damped sum-product belief propagation (200 sweeps, damping 0.5, LLRs clipped at ± 20), computing exclusive products by prefix/suffix passes — a plain $\prod_j t_j/t_i$ divides $0/0$ whenever any message saturates, which silently puts BP at chance; the prefix/suffix form avoids this.

Density regime. At $m/n \le 1$ the constraint system does not determine the labelling and every method, exact or neural, is bounded well below 100: at $m/n = 0.5$ elimination reaches 59.8 and BP 56.7 with only 24.2% of hidden nodes determined; at $m/n = 1.0$, 96.2 and 95.8 against 91.6% determined. We report the identified regime $m/n = 3$ throughout so that the reference is 100 and a shortfall is attributable to the model.

Pairwise co-occurrence does not determine the answer. The two 3-uniform hypergraphs on six nodes $H_+ = \{123, 145, 246, 356\}$ and $H_- = \{124, 135, 236, 456\}$ are two Pasch configurations covering the same pairs (Ling et al., 2000): every node has degree 2 in both and every pair of nodes co-occurs equally often, so $B_+B_+^\top = B_-B_-^\top$ for their incidence matrices $B_\pm$. With the same anchors $y_1 = y_2 = 0$, $y_6 = 1$ and a parity constraint on every hyperedge, each has a unique solution, and the two differ on nodes 3 and 4 (exhaustive enumeration of the 2^6 labellings, `pasch_certificate.py`). Any predictor that sees node features, anchors and the pairwise co-occurrence matrix — a clique expansion under any degree normalisation — returns the same output on both and is wrong on at least one of the two nodes in one of them; an exact solver on the incidence structure is right on both. This is a known configuration, not a new result, and it bounds only models whose input is the pairwise matrix; incidence-based models such as those in this paper see the full structure. It is why HYPERCSP's families are defined on hyperedges rather than on pairs.

B FACTORIAL STUDY

Design: 7 datasets $\times$ 6 hyperedge constructions (original, kNN-5, kNN-10, clique-ego, size-split-5, random-matched) $\times$ 5 architectures (MLP, AllSet, ED-HNN, UniGCNII, and an ordered-scan model[1]) $\times$ 5 seeds, matched width, depth, optimiser and splits. Exactly one cell was absent — the clique-ego construction of Walmart-Trips for the ordered-scan model, which ran out of GPU memory — and that construction is dropped for Walmart-Trips so the design stays balanced; every remaining cell has all 5 seeds before the decomposition; the variance shares are η^2 from a two-way construction $\times$ architecture decomposition with the seed residual as noise. The architecture factor excludes MLP, which has no hyperedge aggregation to vary.

[1]The ordered-scan model sorts each hyperedge's members by a linear score of their features and runs a bidirectional Mamba state-space model over them, fused with a sparse hypergraph convolution and a restart blend. The sort uses detached scores, so the score map receives no gradient and stays at its initialisation; the order changes during training only because the features it scores do.

	variance share (η^2, %)				absolute effect (pp)		
dataset	constr.	arch.	inter.	**seed**	constr. spread	arch. spread	seed SD
Actor	4.6	83.9	1.4	10.1	0.83	3.16	0.40
Twitch-gamers	23.5	1.1	7.7	**67.7**	1.56	0.28	0.81
Amazon-ratings	62.2	4.7	4.7	28.3	1.49	0.51	0.45
Pokec	2.3	2.0	8.9	**86.8**	0.24	0.19	0.49
Citeseer	9.4	37.3	5.0	48.3	1.44	2.43	0.98
Cora-CC	36.2	29.1	3.9	30.8	4.87	4.23	1.47
Walmart-Trips	17.1	26.4	49.2	7.3	1.47	1.71	0.32
mean				40.0	**1.70**	**1.79**	0.70

We report absolute spreads alongside the variance ratios deliberately: the ratio is unstable and misleading where both effects are small. Twitch-gamers gives a construction/architecture variance ratio of $20.6\times$, but the underlying spreads are 1.56 and 0.28 pp against a seed SD of 0.81 — nothing is resolved there. Averaged over datasets the two factors are comparable (1.70 versus 1.79 pp) and the construction exceeds the architecture on 4 of 7 datasets, so we do not claim either dominates; we claim both are small next to the feature path (§3) and next to the seed noise.

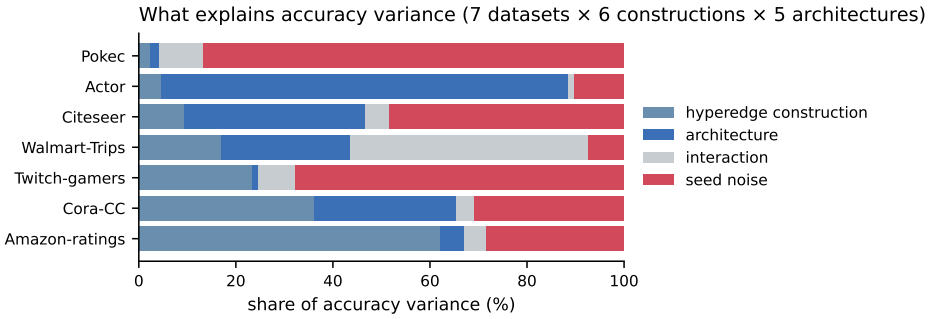

Figure 1: Variance decomposition per dataset. Seed noise is the largest single component on average, and on Pokec and Twitch-gamers it dominates both design factors.

C Detection power of the null

`parity`, $n = 300$, $r = 3$, $m/n = 3$, $\rho = 0.40$, 8 layers, 40 runs per cell, corrected protocol. A fraction of the m hyperedges are constraint-satisfying; the rest are uniform random r-subsets. The plain reference solver treats distractors as valid equations and collapses (it reads $\sim$50 from 50% dilution down), so we report the *oracle* reference — elimination on the constraint hyperedges only, i.e. the fraction of hidden nodes they determine. It is not an accuracy bound: undetermined binary nodes guessed at chance add $(1-q)/2$. The 0% row is the zero-structure control. p is a paired t-test over shared instances.

constr. fraction	oracle (det.)	mean agg.	Deep Sets	ProdSet	Δ vs. mean (p)
100%	100.00	52.80 ± 5.56	65.24 ± 17.77	$\mathbf{86.31 \pm 21.81}$	$+33.52$ (5×10^{-12})
50%	99.0	51.96 ± 4.93	53.39 ± 7.12	60.46 ± 12.04	$+8.50$ (9×10^{-5})
25%	84.2	51.59 ± 3.44	50.63 ± 4.70	50.56 ± 5.01	-1.04 (n.s.)
12%	30.3	50.24 ± 5.01	50.31 ± 4.20	50.28 ± 4.94	$+0.04$ (n.s.)
0% (control)	0	49.57 ± 5.09	51.44 ± 4.93	50.11 ± 4.90	$+0.54$ (n.s.)

Under the corrected protocol the separation is significant at 50% constraint hyperedges and not at 25%; we did not estimate a power curve, so this is the observed range rather than a detection threshold. The zero-structure control shows no significant offset ($+0.54$, $p = 0.42$), which does not prove the offset is zero. (An earlier run under the naive protocol of §8 placed it between 12% and 25%, but with a $+1.5$ pp offset at zero structure; we do not rely on it.) At 50% ProdSet recovers well under half of what the constraints determine.

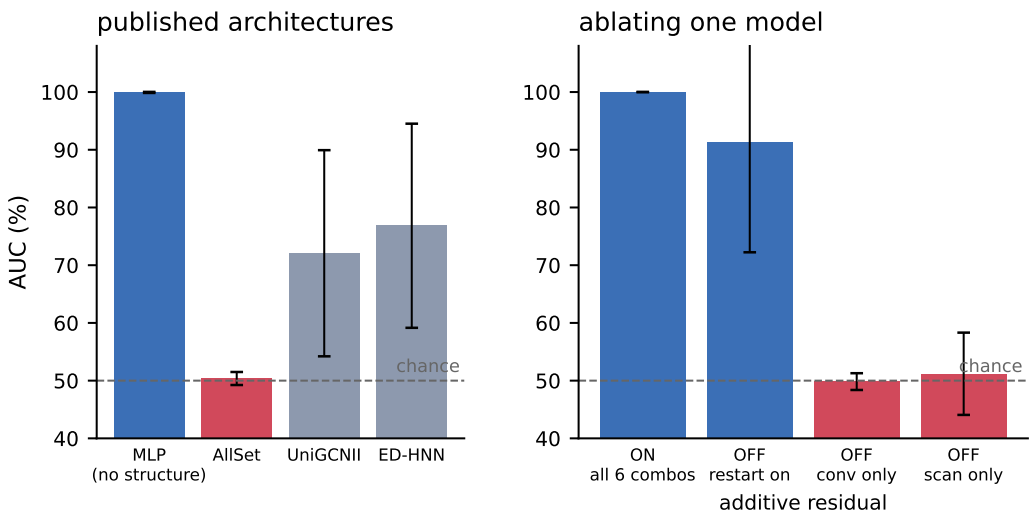

Figure 2: Hypergraph parity with informative features. Left: simplified re-implementations of published architectures separate by how directly they carry the node's own features, not by their aggregation operator. Right: ablating a single model — with an additive residual, six aggregation combinations are identical at 100.00 ± 0.00; without it the aggregation-only variants fall to chance, while the restart-only variant, which keeps a feature path, solves $33/40$. Error bars are one standard deviation, over five seeds on the left and the 40 runs of Table 2 on the right.

Transfer to a real hypergraph. The bound above is measured on a synthetic hypergraph. To test whether it transfers, we inject parity hyperedges consistent with the *real* binary labels into Twitch-gamers ($n = 16{,}812$, real features, real hyperedges retained, real arity distribution) at per-node density m/n, and compare mean aggregation and PRODSET under the real-data protocol (2 layers, dropout 0.5, validation early stopping, 5 seeds). Density is per node because that is what governs identifiability: Twitch-gamers has only 2,627 real hyperedges ($m/n = 0.16$), so injecting "as many as the real ones" leaves the oracle at 14.5 and no detector, exact or neural, can succeed — our first attempt did exactly this and returned an uninformative null.

protocol	inj. m/n	oracle	MLP	mean	PRODSET	Δ (paired p)
standard (labels in loss)	2.00	99.80	50.53	50.58	50.83	+0.25 (n.s.)
anchored	0.00	0.00	50.57	51.19	50.74	−0.45 (n.s.)
anchored	0.25	25.82	50.57	50.78	61.73	+10.95 (9×10^{-6})
anchored	0.50	63.57	50.57	50.59	74.66	+24.06 (10^{-6})
anchored	1.00	94.31	50.57	50.25	91.33	+41.09 (4×10^{-8})
anchored	2.00	99.80	50.57	50.60	**99.38**	+48.78 (3×10^{-9})

Two things transfer and one does not. Under the anchored protocol of HYPERCSP — the same code as Table 5: labels of training nodes as input, a fresh 30% withheld and scored each epoch, checkpoint selected on validation — PRODSET approaches the oracle on a real hypergraph as density grows (99.38 against 99.80 at $m/n = 2$) and separates from mean aggregation down to $m/n = 0.25$, where the injected hyperedges are 62% of the total; there is no significant offset at zero structure (-0.45). Counting undetermined nodes at chance, it realises 91% and 78% of the gain over chance that the oracle permits at $m/n = 0.25$ and 0.5. Mean aggregation stays at chance throughout. The per-run mask sizes and their pairwise overlaps (all zero) are saved with the results. What does *not* transfer is the standard protocol: with labels entering only through the loss, PRODSET exploits none of the structure (50.83 against an oracle of 99.80). A constraint propagator needs anchors to propagate from, and the standard node-classification protocol gives it none — so under that protocol the power to detect constraint structure is essentially zero *regardless of what the data contains*. This is why Table 4 is reported under both protocols.

D RE-IMPLEMENTED OPERATORS, TUNED PER ARM

Table 6: Re-implementations of the published operators (scatter-mean ED-HNN, linear-projection AllSet, simplified UniGCNII — not the released models, unlike Table 1) and ours, each at its own validation-selected configuration ($L \in \{2, 4\}$, lr $\in \{0.01, 0.003\}$; 10 seeds at the selected setting). Our mean aggregation arm sits inside the band these re-implementations span on all three; the released models of Table 1 are the stronger reference for the field itself. Tuning does not rescue PRODSET: it is never the best arm and its selected depth is always 2. On Citeseer and Actor the structure-free MLP beats most hypergraph operators.

dataset	MLP	AllSet	ED-HNN	UniGCNII	mean agg.	PRODSET
Citeseer	$\mathbf{72.69} \pm 1.45$	70.93 ± 1.79	71.58 ± 1.54	72.32 ± 1.45	72.66 ± 1.47	71.93 ± 1.66
Cora-CA	75.35 ± 1.32	82.05 ± 1.70	83.12 ± 1.19	83.31 ± 1.28	$\mathbf{83.49} \pm 1.08$	82.63 ± 1.06
Actor	83.91 ± 0.63	83.21 ± 0.45	$\mathbf{84.61} \pm 0.51$	84.38 ± 0.46	82.34 ± 0.54	81.86 ± 0.59

E COST

Table 7: Cost of the aggregator at matched settings (2 layers, width 64), one NVIDIA RTX A5000 (24 GB), PyTorch 2.10 / CUDA 12.8, mean over 12 timed epochs after 3 warm-up: milliseconds per training epoch and peak memory, mean aggregation versus PRODSET. Incidences (non-zeros of the incidence matrix) are given because the product is evaluated per arity group. The slowdown is not explained by the maximum arity alone (Twitch-gamers, maximum 41, is slower than Pubmed, maximum 171): it also depends on how many distinct arity groups must be launched and how large they are. Part is implementation (a fused kernel would amortise the grouping) and part intrinsic (a product needs prefix and suffix passes where a sum needs one).

dataset	incidences	max arity	mean agg. (ms)	PRODSET (ms)	peak MB (mean / PRODSET)
Citeseer	3,453	26	6.1	189.4 (31×)	126 / 141
Cora-CC	4,786	5	5.0	49.7 (10×)	60 / 80
Cora-CA	4,585	43	4.3	310.2 (72×)	60 / 79
Pubmed	34,629	171	5.8	662.9 (115×)	179 / 321
Actor	55,134	30	5.1	261.0 (51×)	127 / 349
Amazon-ratings	6,479	6	3.5	60.0 (17×)	100 / 144
Twitch-gamers	16,376	41	3.8	680.4 (177×)	77 / 149

F HYPERPARAMETERS

All HYPERCSP runs: width 32, Adam, learning rate 0.01, 400 epochs, gradient clipping at 1.0, no dropout, residual update $h \leftarrow h + \mathrm{act}(\mathrm{agg}(h))$. Corrected protocol: 30% of anchors withheld and scored per epoch; hidden nodes split $25/75$ into validation (checkpoint selection by best validation accuracy) and test (reported). Depth is 8 except in Table 8 and Appendix G. All real-benchmark runs: width 64, 2 layers, dropout 0.5, Adam, learning rate 0.01, weight decay 10^{-4}, 500 epochs with early stopping at patience 100 on validation accuracy, splits $50/25/25$ seeded per run. The per-arm search of Appendix D used 300 epochs with patience 60 for both the search and the final runs.

G PROTOCOL ROBUSTNESS, SUPERVISION AND DEPTH

Supervision curve. `parity`, $n = 300$, $m/n = 3$, 8 layers, corrected protocol, 40 runs per cell (the 0.25, 0.40 and 0.60 rows are those of Table 3). Solved counts, with mean accuracy in parentheses.

ρ	mean agg.	Deep Sets	PRODSET
0.25	0/40 (50.6)	0/40 (51.6)	23/40 (78.6)
0.30	0/40 (50.6)	0/40 (55.5)	22/40 (79.1)
0.35	0/40 (51.4)	0/40 (59.6)	24/40 (78.7)
0.40	0/40 (52.6)	0/40 (63.5)	24/40 (82.4)
0.50	0/40 (55.7)	3/40 (77.1)	28/40 (88.1)
0.60	0/40 (59.8)	21/40 (86.0)	30/40 (91.4)

The mean arm never solves a run. Deep Sets' average rises steadily with ρ while its solve rate stays at zero until $\rho = 0.50$ (3/40), then reaches 21/40 at $\rho = 0.60$; PRODSET's rate is flat at $\sim$60% from $\rho = 0.25$ to 0.40 and rises to 70–75% at $\rho \geq 0.50$.

Table 8: Ablations on `parity`, $\rho = 0.40$, corrected protocol, 40 runs. Left: the aggregator ladder at $L = 4$. Right: weight-tied variants at $L = 16$, one shared rule applied 16 times.

variant ($L = 4$, untied)	solved	variant ($L = 16$, tied)	solved
mean aggregation	0/40	no damping, no renorm.	9/40
exclusive product	36/40	+ damping, LayerNorm	**40/40**
+ learned temperature	38/40	+ damping, LN, signed	37/40
+ near-identity init	37/40		

Depth. The same cell at $L = 4$ instead of 8, every arm, corrected protocol, 40 runs:

family ($\rho = 0.40$, $L = 4$)	mean	sum	excl. sum	Deep Sets	PRODSET
`parity`	0/40	0/40	0/40	1/40	**37/40**
`mod3`	0/40	0/40	0/40	0/40	**38/40**

At $L = 4$ the four additive arms solve 1 of 320 runs on the modular families (one Deep Sets run on `parity`) and PRODSET 75 of 80. Depth changes the product's rate, not the separation, so the $L = 8$ comparison of Table 3 is conservative for PRODSET.

Controls for the operation. Modular cells of Table 3, solved of 40; p is paired exact McNemar against PRODSET.

family	ρ	Deep Sets	excl. Deep Sets	excl. mean (pipeline)	product via power sums	PRODSET
`parity`	0.25	0	0 (2×10^{-7})	0	17 (0.24)	23
	0.40	0	21 (0.65)	0	21 (0.65)	24
	0.60	21	27 (0.63)	6	30 (1.00)	30
`mod3`	0.25	0	0 (8×10^{-6})	0	18 (1.00)	18
	0.40	0	1 (3×10^{-6})	0	27 (0.42)	23
	0.60	4	13 (5×10^{-4})	0	29 (0.75)	31
pooled		25/240	62/240	6/240	142/240	149/240

Sensitivity to the solve criterion. The six modular cells of Table 3, pooled, at three accuracy thresholds:

criterion	linear additive (mean, sum, excl. sum)	Deep Sets	PRODSET
$\geq 90\%$	0/720	44/240	163/240
$\geq 95\%$	0/720	25/240	149/240
$\geq 99\%$	0/720	3/240	81/240

The failure of the linear arms does not depend on the criterion; how completely PRODSET solves the task does. "Solved" in this paper means $\geq 95\%$ on the test nodes, not exact recovery.

Loss mode × selection rule. The same runs scored both ways (solved of 40; `parity` and `onehot`, $\rho = 0.40$, $L = 8$):

		parity			onehot		
loss	selection	mean	Deep Sets	PRODSET	mean	Deep Sets	PRODSET
withheld anchors	validation	0	3	24	34	35	37
withheld anchors	best test epoch	0	3	24	40	40	39
copyable	validation	0	0	40	0	0	39
copyable	best test epoch	0	0	40	0	0	40

How stable is a solve rate? Individual runs are not reproducible across processes (§8), so each 40-run cell is itself a noisy estimate. Three cells were run twice, in independent sweeps with identical settings: PRODSET on `parity`, $\rho = 0.40$, $L = 8$ gave $24/40$ and $28/40$; at $L = 4$ it gave $37/40$ both times; the mean arm at $L = 4$ gave $0/40$ both times (52.56 and 52.54). Under the naive protocol, three re-estimates of $\rho = 0.25$ cells moved by 5 points on `parity` and 10 on `mod3`. We therefore do not interpret solve-rate differences below about 10 points. The conclusions of §5–6 rest on differences of 18–37 runs out of 40.

H FEATURE-PATH ABLATION ACROSS ARCHITECTURES AND DATASETS

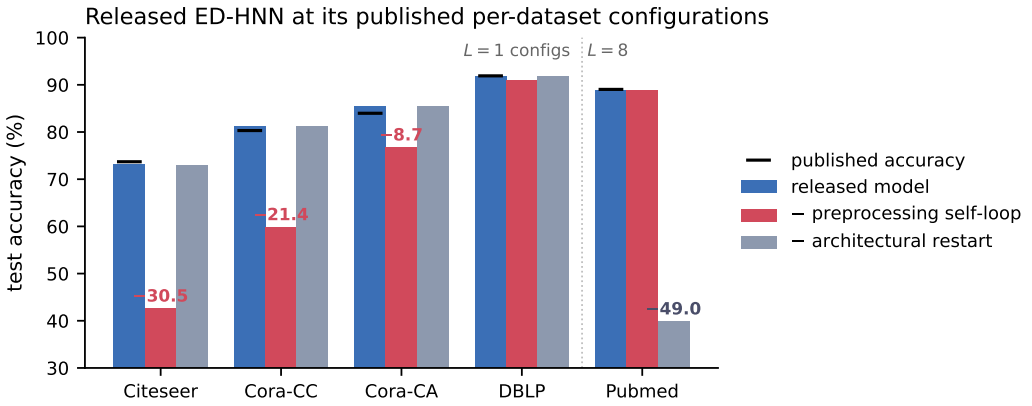

Figure 3: The released ED-HNN implementation at its published per-dataset hyperparameters; black dashes mark the accuracy reported in the original paper. For the four L=1 configurations the authors disable the restart, so the preprocessing self-loop is the only node-feature path and deleting it costs up to 30.5 pp; deleting the (absent) architectural path costs nothing. The one L=8 configuration inverts this exactly.

The ablation of §3 repeated on four architectures — attention convolution (AllSet), scatter-mean $V \to E \to V$ (ED-HNN), initial-residual propagation (UniGCNII), and an ordered state-space scan — across six datasets, 5 seeds, matched width (64), depth (2), optimiser and splits. -PATH removes every node-feature path (residual, restart and the node term of the concatenation) with parameter shapes unchanged. Wherever a model learns at all (Citeseer, Cora-CC, Actor, Pokec) the cost is 6–32 pp and the four architectures land within a narrow band of one another; on Twitch-gamers and Amazon-ratings nothing learns and nothing is lost. Removing the path drops every model *below* the structure-free MLP wherever the MLP is above chance.

dataset	architecture	FULL	-PATH	cost	MLP	-PATH − MLP
Citeseer	AllSet	70.53	41.16	+29.37	71.64	−30.48
Citeseer	ED-HNN	71.69	40.07	+31.62	71.64	−31.57
Citeseer	UniGCNII	72.39	41.40	+30.99	71.64	−30.24
Citeseer	ordered scan	73.14	41.06	+32.08	71.64	−30.58
Cora-CC	AllSet	75.21	57.55	+17.67	73.71	−16.16
Cora-CC	ED-HNN	77.78	57.22	+20.56	73.71	−16.48
Cora-CC	UniGCNII	78.08	58.79	+19.29	73.71	−14.92
Cora-CC	ordered scan	79.76	58.97	+20.80	73.71	−14.74
Actor	AllSet	82.97	62.68	+20.29	86.06	−23.38
Actor	ED-HNN	83.83	62.70	+21.13	86.06	−23.36
Actor	UniGCNII	83.92	62.23	+21.68	86.06	−23.82
Actor	ordered scan	86.32	64.17	+22.15	86.06	−21.89
Pokec	AllSet	59.68	52.27	+7.41	60.07	−7.80
Pokec	ED-HNN	59.52	51.67	+7.85	60.07	−8.40
Pokec	UniGCNII	59.16	52.81	+6.35	60.07	−7.27
Pokec	ordered scan	59.43	51.97	+7.46	60.07	−8.10
Twitch-gamers	AllSet	50.93	49.63	+1.30	51.43	−1.80
Twitch-gamers	ED-HNN	51.03	50.04	+0.99	51.43	−1.38
Twitch-gamers	UniGCNII	51.44	50.99	+0.46	51.43	−0.44
Twitch-gamers	ordered scan	51.53	50.01	+1.52	51.43	−1.42
Amazon-ratings	AllSet	28.96	28.49	+0.47	27.64	+0.85
Amazon-ratings	ED-HNN	29.18	28.48	+0.70	27.64	+0.84
Amazon-ratings	UniGCNII	29.12	28.49	+0.63	27.64	+0.85
Amazon-ratings	ordered scan	29.10	28.49	+0.61	27.64	+0.85

I WHY THE RESTART MATTERS ON PUBMED: DISAMBIGUATION

Table 1 shows the restart worth 49.0 pp on Pubmed and ≤ 0.15 pp elsewhere. Three candidate explanations, tested on the released code with the restart enabled ($\alpha = 0.5$) and self-loops present. Only parameterised depth reproduces the effect. The first two rows hold L at 8 — Pubmed's own propagation depth — and the restart is still worth nothing there; raising the per-layer MLP depth at that same L reproduces Pubmed's -49 pp exactly.

The comparison is within the published shallow-MLP configuration, and the restart's value is configuration-dependent: at other settings (no normalisation, larger learning rate) we observe it becoming necessary at $L \geq 4$ even with MLP $0/0/1$, which is the ordinary deep-network optimisation effect rather than anything hypergraph-specific. The claim here is the narrow one — propagation depth alone does not explain Pubmed — and it rests on the same-L rows below.

candidate	setting	FULL	-SKIP	Δ
propagation depth	Citeseer, L=8, MLP $0/0/1$	69.35	70.02	+0.67
	Cora-CC, L=8, MLP $0/0/1$	77.58	77.49	−0.09
normalisation	Pubmed, L=8, LayerNorm for `None`	44.93	43.03	(model broken)
parameterised depth	Citeseer, L=8, MLP $2/2/2$	70.37	21.14	**−49.2**
	Cora-CC, L=8, MLP $2/2/2$	73.17	29.35	**−43.8**
(published Pubmed)	Pubmed, L=8, MLP $2/2/2$	88.92	39.89	−49.0

J DATASETS

Real benchmarks used in §3 and §7. Citeseer, Cora-CC, Cora-CA, Pubmed and DBLP are the AllSet co-citation/co-authorship releases; Actor, Amazon-ratings, Twitch-gamers and Pokec are the heterophilic hypergraph releases of Li et al. (2025), who report that existing HNNs struggle on them. Arity is the hyperedge size.

dataset	nodes	features	classes	hyperedges	arity mean	arity median	arity max
Citeseer	3,312	3,703	6	1,079	3.2	2	26
Cora-CC	2,708	1,433	7	1,579	3.0	3	5
Cora-CA	2,708	1,433	7	1,072	4.3	3	43
Pubmed	19,717	500	3	7,963	4.4	3	171
DBLP	41,302	1,425	6	22,363	4.5	3	202
Actor	16,255	50	3	10,164	5.4	5	30
Amazon-ratings	22,299	111	5	2,090	3.1	3	6
Twitch-gamers	16,812	7	2	2,627	6.2	6	41
Pokec	14,998	65	2	2,406	2.3	2	7

Hyperedges per node (m/n) ranges from 0.09 (Amazon-ratings) to 0.63 (Actor); every benchmark is below 1. On HYPERCSP this quantity governs identifiability (Appendix A): at $m/n = 0.5$ only 24.2% of hidden nodes are determined even by exact elimination, and 91.6% at $m/n = 1.0$. This is one concrete reason constraint-type structure, if present, would be weak on these benchmarks.

K FULL AGGREGATOR ABLATIONS

Every cell of the damping and renormalisation study (`parity`, $\rho = 0.40$, 40 runs, solve rate at ≥ 95; Fisher exact p against the row's baseline). **These cells were measured under the naive protocol of §8** (loss on anchors whose labels are inputs, best test accuracy over epochs), whose absolute rates are inflated; they are kept as a record of relative effects under that protocol only. The tied cells that enter Table 8 were re-run under the corrected protocol.

setting	variant	solved	vs. baseline p
untied, $L = 8$	baseline (no damping, no renorm., `relu`)	$33/40$	—
	+ damping 0.5, LayerNorm, `relu`	$18/40$	0.001
	+ damping 0.5, LayerNorm, signed	$14/40$	< 0.001
	+ damping 0.5, clamp, signed	$34/40$	1.00
untied, $L = 32$	none (plain PRODSET: the depth test of §6)	$31/40$	1.00
	+ damping 0.5, clamp, signed	$27/40$	0.20
untied, $L = 8$	baseline, $3\times$ training budget	$26/40$	0.13
tied, $L = 16$	baseline (no damping, no renorm., `relu`)	$3/40$	—
	+ damping 0.5, LayerNorm, `relu`	$20/40$	< 0.001
	+ damping 0.5, LayerNorm, signed	$\mathbf{32/40}$	< 0.001
	+ damping 0.5, clamp, signed	$6/40$	0.48
	+ damping 0.5, clamp, `relu`	$19/40$	< 0.001

"Signed" removes the `relu` on the message path so that messages keep their sign, as belief-propagation LLRs do. Without renormalisation this collapses at depth (untied $L = 16$ and $L = 32$ signed: 53.7 and 54.3%, every run at chance), which is why `relu` is the default despite the sign it discards: it is acting as the squashing that keeps an iterated product bounded, the role damping and renormalisation fill in BP.

L REPRODUCIBILITY OF INDIVIDUAL RUNS

The measurements behind §8. Left: one command re-run three times on the GPU (v4, $L = 32$, seeds 0–9, mean accuracy). Right: seed 0 of the same configuration on the CPU, three repeats each with default BLAS threading and single-threaded. These runs used the naive protocol of §8; the non-reproducibility is a property of the arithmetic, not of the protocol, and the corrected-protocol replications of Appendix G show it too.

GPU repeat	mean $\pm$ SD over seeds 0–9	CPU, seed 0	three repeats
1	84.76 ± 20.04	10 BLAS threads	$55.56,\ 56.00,\ 100.00$
2	93.33 ± 13.56	`OMP_NUM_THREADS=1`	$55.56,\ 55.56,\ 55.56$
3	99.82 ± 0.53		

Within one process everything is bit-reproducible: generated instances, initial weights, and repeated training runs are identical. The divergence is across processes, and disappears with a single BLAS thread, which fixes the floating-point reduction order.

M LDPC DECODING

$(3, 6)$-regular Gallager code, $n = 300$ bits, $m = 150$ checks, rate $1/2$. Codewords are drawn uniformly from the null space of H and sent through a binary symmetric channel with flip probability p; the code's BSC threshold is ≈ 0.084, so all three settings are below it. Node features are the received bit (one-hot) and a noise channel; the label is the transmitted bit. 40 training, 10 validation and 10 test codewords are stacked as disjoint components of one hypergraph; BP, hard decision and every network are scored on the same 10 test codewords; 8 layers, width 64, no dropout, validation early stopping, 10 seeds, otherwise the real-data harness of §7. Welch p is PRODSET against the best additive arm. Hard decision is the floor (accuracy $1 - p$); 50-sweep damped sum-product BP with the exact channel LLRs is the reference decoder.

p	hard decision	BP (reference)	MLP	mean	sum	Deep Sets	PRODSET	Welch p
0.03	96.88	100.00	96.88 ± 0.30	96.87 ± 0.29	96.86 ± 0.27	96.92 ± 0.30	$\mathbf{99.69 \pm 0.18}$	3×10^{-13}
0.05	94.98	99.94	94.97 ± 0.33	94.98 ± 0.34	$89.84 \pm 12.99^{\dagger}$	95.04 ± 0.41	$\mathbf{98.30 \pm 0.57}$	2×10^{-10}
0.07	92.84	98.36	92.83 ± 0.45	92.83 ± 0.46	92.84 ± 0.47	92.82 ± 0.47	$\mathbf{95.24 \pm 1.00}$	2×10^{-5}

†One of ten sum-aggregator runs diverged to chance (51.4); the other nine are at hard decision. Share of BP's error reduction recovered by PRODSET, $(e_{\text{hard}} - e_{\text{PRODSET}})/(e_{\text{hard}} - e_{\text{BP}})$, all on the same 10 test codewords: 90%, 67%, 43% (per-seed averages 90, 68, 44%; paired t against the best additive arm $p \leq 6 \times 10^{-7}$).

Cost against BP. On the same GPU, decoding one $n = 300$ codeword takes 23.7 ms for 50 sweeps of vectorised sum-product BP, and 4.8 ms for an 8-layer PRODSET forward pass (7.2 ms for mean aggregation). Training PRODSET costs 78.3 ms per epoch against 9.7 for mean aggregation (8.1×, at $N = 18{,}000$, arity 6), i.e. about 25 s per seed; BP needs no training and is more accurate at every p. PRODSET is therefore a diagnostic here, not a competing decoder: it shows that a generic hypergraph layer uses the check structure only when its aggregation is multiplicative. The gap to BP widens toward the threshold, where BP needs its full 50 sweeps and 8 layers are fewer iterations than that. At $p = 0.10$, above threshold, BP gains only 2.4 pp over hard decision, so that setting is uninformative and is not reported. Neural BP decoders (Nachmani et al., 2016; 2018) improve on BP for short linear codes by attaching learned weights to a BP unrolling; the point here is not to compete with them but that a generic hypergraph layer only does so when its aggregation is multiplicative.

N MESSAGE MAGNITUDE OF THE EXCLUSIVE PRODUCT ON REAL FEATURES

Mean $|\prod_{v \neq u} t_v|$ over members and channels, by hyperedge arity, for PRODSET's first aggregation layer on real benchmarks (seed 0, standard protocol), at initialisation (near-identity in/out maps) and, for Cora-CA, after training. "After training" means the final weights of a run with 200 epochs and patience 60, not the validation-selected checkpoint of Table 4; it is an illustration, not a measurement at the reported model. On HYPERCSP the anchor one-hots put $|t_v|$ near 1 and the product stays large; on continuous real features $|t_v|$ is small everywhere and the product shrinks by roughly a factor of 50 per member, to 10^{-6}–10^{-8} at arity 4–5. These values are far above float32's smallest normal number ($\sim 10^{-38}$), so this is not numerical underflow; the messages are small relative to the residual path they are added to, which is enough for the operator to contribute little. We did not run an intervention (e.g. rescaling messages) that would show smallness to be the *cause* of the real-data null; the observation only means that the null cannot be read as evidence about the data. After training, arity-5 messages reach 2×10^{-2} on Cora-CA — the one benchmark where the hypergraph is worth 8.7 pp over an MLP — but stay at 5×10^{-5} on Pubmed, 2×10^{-4} on Actor and 1×10^{-6} on Twitch-gamers. Full per-arity values are in the released `nextdir_phase10_underflow.json`.

arity	Pubmed	Actor	Twitch-gamers	Cora-CA (init → trained)
2	6.6×10^{-3}	2.7×10^{-2}	5.6×10^{-2}	$1.2 \times 10^{-2} \to 2.5 \times 10^{-1}$
3	1.2×10^{-4}	1.5×10^{-3}	7.0×10^{-3}	$2.1 \times 10^{-4} \to 9.1 \times 10^{-2}$
4	2.3×10^{-6}	9.6×10^{-5}	9.6×10^{-4}	$4.0 \times 10^{-6} \to 4.0 \times 10^{-2}$
5	4.6×10^{-8}	7.5×10^{-6}	1.4×10^{-4}	$8.7 \times 10^{-8} \to 2.4 \times 10^{-2}$
8	4.6×10^{-13}	6.9×10^{-9}	5.7×10^{-7}	$7.8 \times 10^{-13} \to 4.6 \times 10^{-3}$
10	2.4×10^{-16}	7.8×10^{-11}	1.7×10^{-8}	$2.0 \times 10^{-16} \to 2.7 \times 10^{-3}$

O SOFTWARE AND REPRODUCTION

Every table is produced by a script in the supplementary material, whose README maps scripts and raw per-run files to tables. HYPERCSP: `nextdir_phase3_generality.py` (Tables 3 and 8, Appendices C and G; corrected protocol by default) with models in `nextdir_phase2_stability.py` and statistics in `analyze_masked.py`; references in `nextdir_phase1_boundary.py` and `nextdir_phase7_oracle.py`. Real data: `nextdir_phase4_realdata.py` (Tables 4 and 5), `nextdir_phase5_cost.py`, `nextdir_phase6_baselines.py`, `nextdir_phase8_inject.py`, `nextdir_phase9_ldpc.py` and `nextdir_phase10_underflow.py`. Feature paths: `released_code_ablation.py` (Table 1; requires the public ED-HNN repository), `feature_path_attribution.py` (Table 2), `factorial_study.py` with `analyze_factorial.py` (Appendix B) and `residual_ablation_realdata.py` (Appendix H). Raw files of the naive protocol of §8 are included separately and labelled as such. On CPU, `OMP_NUM_THREADS=1` reproduces an individual run bit-exactly; GPU runs are not bit-reproducible across processes, and the reported quantities are rates over 40 runs for that reason.