XE4-S: Dense Composition over Thirty-One Specialists after a Routing Collapse at $N = 3$

Thomas Garren **Sophia**

SOPHIA XT LLC

sophiaxt.com

Abstract

Our previous architecture selected among specialists rather than combining them. It composed NCN Module Specialists, rank-16 LoRA deltas of about 16.7 MiB shipped as `.ncn` artifacts carrying a capability capsule, an explicit scope and exclusion list, a bound validation set and a provenance manifest, and it chose a sparse active set of them by closed-form GSVD capsule routing. It worked at $N = 2$ and collapsed past it. Top-1 route accuracy over a 30-prompt held-out set was 100%, 23.3%, 36.7%, 36.7% and 46.7% for N of two through six; the `code` specialist became a dead attractor at 0% from $N = 5$; a learned replacement router sent every prompt to one node at exactly 0.0 bits of mutual information; and expanding one specialist's corpus nearly twentyfold made $N = 6$ worse rather than better. That program was specified, measured and published as a failure [1].

XE4-S is what we built instead. It composes thirty-one specialists densely, selects nothing, and takes its routing supervision from a corpus-side label channel written at build time. At $N = 31$ it routes 18 of 19 task shapes correctly in distribution and assigns the twelve specialists it never trained exactly 0.000 weight, mean and max, on every probe, with no mechanism asking it to. Both sides of that comparison are the same lab's work on its own record, and the failing side was written down first.

The rest of the paper is the model, and the model does not work. XE4-S is 130.6 M parameters trained from scratch on one RTX 3090 with no pretrained base, in five stages, over three corpora. Its English is fluent and accurate. On 178 held-out probes it reaches 57.3% tool-choice exact match against a dense monolith's 82.6%, and on real-user phrasings of the same tasks it reaches 62% against that baseline's 100%. Exact argument reproduction is 0%. So is the baseline's, once the probe set stops using invented proper nouns that the baseline's own tokenizer had memorised: its apparent 81.4% measured a tokenizer, not a model. Stored memory does not change behaviour, `memory_get` being emitted 31 times whether the fact was written or never written. Growth by use moved held-out accuracy by +0.0 over 300 interactions. Output novelty collapses to 0.00 by token 144 and the model serves at 4.6 tok/s. Truncating the bank from thirty-one specialists to eight is free, 73.0% against 71.9%, which retracts an earlier claim of ours that it cost 21 points; that figure was our own selection-rule defect. Four separate bugs found in one session belong to one class: state that is not a fixed-shape buffer, silently not kept in step with what it indexes.

One diagnosis covers nearly all of it. The training corpus is template-generated, the specialists fit it closely, and what they learned was the templates rather than the behaviours. The architecture has not been shown to fail. What is missing is training that makes using any of it necessary.

1 Introduction

The standard way to hold many capabilities in one small model is a mixture of experts [2, 3]. A router reads the input, picks one expert of many, and only that expert runs. Total parameters grow and active parameters do not. The mechanism that makes it cheap is selection, and selection is the part that broke for us.

Our previous architecture, the NCN Module Specialist line [1], packaged each capability as a narrow parameter-efficient delta over a frozen pretrained base: rank-16 LoRA at about 16.7 MiB, rank-8 at about 8.4 MiB, written out as an `.ncn` artifact with a capability capsule, an explicit scope and exclusion list, a bound validation set and a provenance manifest that the router consumed at inference. Routing was closed form. A pooled query embedding was scored against the per-specialist capsules by cosine similarity or by a GSVD-routed capsule similarity, and the highest scorers formed a sparse active set. The ship gate was 85% top-1 route accuracy.

At two specialists it was clean: 18 of 20 across bases, with both misroutes low-margin general-to-architecture confusions on the cross-base wrapper. At three and above, over capsules from similar domains, it collapsed, and stayed collapsed however the mechanism was varied. Section 7 gives the whole curve and the three follow-ups that sharpened rather than softened it.

XE4-S changes two things and keeps everything else. It never selects: all thirty-one specialists receive non-zero weight and their outputs are summed. And it never learns to route from embedding geometry: one uint8 per token, written when the corpus is built, both masks the loss that trains specialist i and supervises the composer that weights it. There is no top-k in this system and therefore no load-balancing loss anywhere in this work.

On top of that sits the property the architecture is named for. The mixture weights are computed from the input and from memory,

$$w = \mathrm{compose}(x, m) \tag{1}$$

where m summarises where a beam descent through a runtime-written memory of user facts terminated. Because m

Preprint. Under review.

depends on stored state that the input does not determine, a fixed input does not fix the weights. Retrieval-conditioned routing is not new [8, 9], and Section 10 says so plainly; what is unusual here is that the retrieved object is user content rather than a cached routing decision, and that the output is dense rather than sparse.

This paper reports one architectural result and a long list of failures. The result is that the collapse which stopped the previous architecture does not appear here at fifteen times the specialist count. The failures are almost all one failure, and Sections 5 and 11 name it: a corpus built from templates teaches templates. Sections 2 to 4 are the system, its curriculum and its data; Sections 5 and 6 are the evaluation and the ablations; Section 7 is the comparison the paper is built around; Sections 8 to 10 are the bugs, the limits and what we decline to claim.

2 Architecture

2.1 The defining equation

A mixture of experts computes $w = \text{gate}(x)$. XE4-S computes Equation 1. Two consequences follow and neither is decorative. Every specialist receives non-zero weight, so there is no active set to choose and nothing that has to learn to choose it. And the descent prints as a path, so the region of memory an answer was composed under can be read off after the fact.

2.2 The stack

Figure 1 is the data path. Tokens enter the trunk. The trunk's hidden states go three places: they are queried against the memory tree, they drive the composer, and they are the input to every specialist. The composer emits thirty-one weights, the specialist outputs are scaled and summed, the sum passes through the retrieval stage, and the head decodes it.

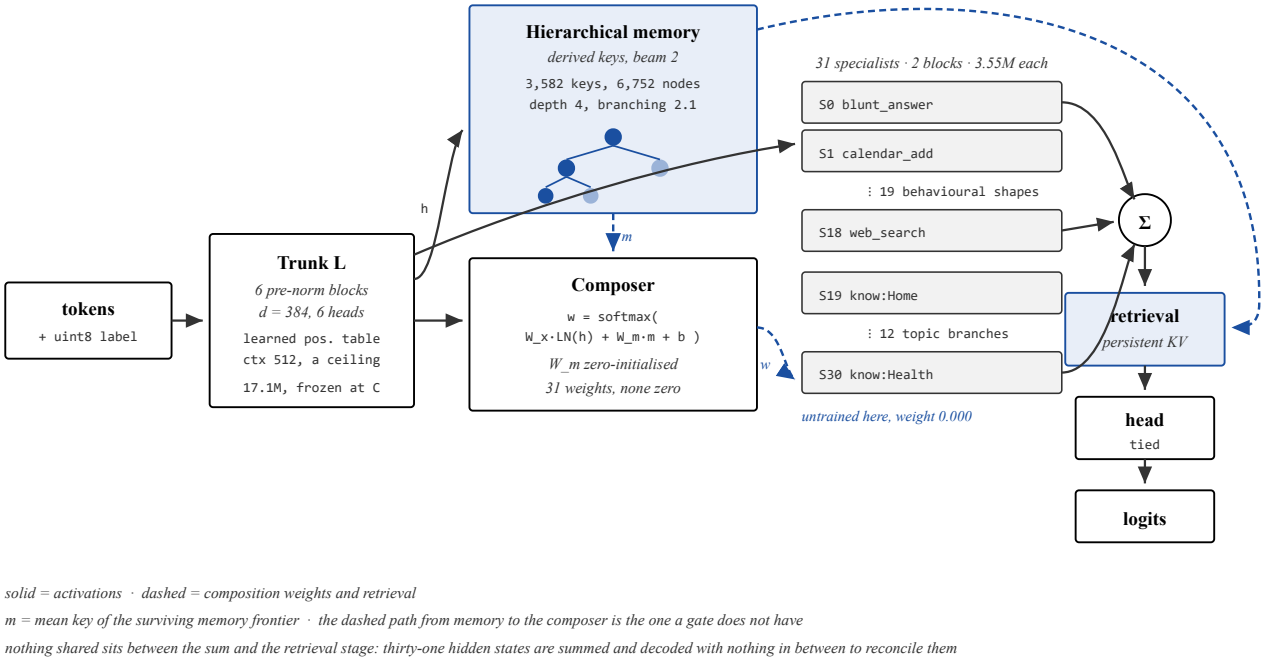


Figure 1: The XE4-S data path. Position enters the trunk as a lookup in a learned table with one row per index, so 512 is a hard ceiling (Section 2.3). The memory is read twice on one forward pass, once as the context vector m that conditions the composition, and once as the retrieval stage that runs after the bank. Nothing shared sits between the sum and that stage: thirty-one hidden states are added and decoded with no layer in between to reconcile them, which Section 11 returns to. The twelve topic specialists are present in this checkpoint and untrained, because XE4-S ran with no knowledge corpus.

Table 1 is the parameter budget. The two totals in it are the two costs this architecture separates. Capability lives in the specialist count; width and trunk depth set what has to be resident to answer a turn.

Table 1: Parameter budget, measured rather than estimated. Our documents carried two totals, 127.5 M and 130.6 M, and both are right under the convention each was using. Reloading the checkpoint and counting settles it: *model.parameters()* returns 130.57 M, of which 2.59 M is the per-fact memory values, leaving 127.98 M of architecture. A further 5.19 M of derived keys and latent slots are buffers and are not parameters.

Component	Params	Note
Trunk, 6 blocks and embedding	17.1 M	trained once, stage A
Specialist, each	3.55 M	×31 = 110.0 M
Output head	6.29 M	tied to token emb.
Composer	24,638	two linear maps
Memory query projections	0.15 M	not a specialist
Memory value vectors	+384 / node	grows when written to
MemK scorer	129	a submodule of the model
Bank total, 31 specialists	127.5 M	all resident as trained
Evaluated checkpoint	130.6 M	6,752 memory nodes written

2.3 Trunk and positions

Six pre-norm decoder blocks [17] at $d = 384$, six heads of width 64, fused bias-free QKV, a $4d$ GELU feed-forward [22], and a head weight-tied to the token embedding [21].

Position enters as a learned embedding table with one row per index. That is a design limit worth stating rather than burying, because it is the reason the context window is exactly 512 and cannot be moved without retraining: there is no row for position 513, and asking for one raises an index error rather than degrading. Everything reported in this paper was measured inside that window, and the memory-contingent evaluation of Section 5.4 exists precisely because the interesting questions about an assistant start outside it.

Stage A trains the trunk through a single specialist path rather than through the full mixture. Evaluating the mixture routes the trunk's gradient through thirty frozen randomly-initialised specialists and dilutes the signal to one thirty-first of the output, so the trunk would have been optimising to feed noise. The single path runs at 0.053 s per step against 0.324, a 6.1× difference. From stage C onward the trunk is frozen, which is what makes every later capability additive rather than a renegotiation of what is already there.

2.4 The memory

A tree. Node keys are the l2-normalised mean of the trunk's token embeddings for the words of the node's breadcrumb. No trained parameter appears in that formula, so inserting a fact cannot move the key of any existing fact; growth without disturbance is a property of the construction rather than a promise about optimisation.

The tree is read three ways, and the third is the one that makes it part of the network rather than a retrieval sidecar. A beam descent with a margin gate produces the context vector m and a printable path, two candidates surviving per level, because a hard top-1 first hop was what made an earlier hierarchical router lose to a flat scan. Each hop contributes a log-probability and the hops compose along the route, so a survivor's weight is the product of its hops; on a five-fact tree that gives 0.868 against 0.132 where a flat mean gave 0.500 against 0.500. And retrieved nodes are prepended to the keys and values that every block attends over. They are not sequence positions, so the causal mask covers only the sequence part and the memory block is left open. An ordinary key-value cache grows with the conversation; this grows with the number of facts, and a fact stays attendable after the turn that produced it has left the window.

Each node owns a learned value vector, so writing facts adds parameters: two facts added five nodes and 1,920 parameters. Each node also owns a latent slot, written by compressing the hidden states of the turn that touched it, because the symbolic tree stores only what survives extraction into a clean breadcrumb and loses nuance, relation and the shape of how something was said. The node's value is a learned prior plus the written latent.

The write is gradient-free and sub-millisecond, addressed by breadcrumb, rewriteable at that address and retractable. Writing moves the output by 7.7×10^{-2}, rewriting the same address a further 1.17×10^{-1}, and forgetting returns the model to baseline at exactly $0.000\times10^{+00}$ with the symbolic path intact. A wrong fact is replaced at its address rather than argued down by later evidence, which a gradient memory cannot do. Larimar [10] reaches three of those four properties by a different route and has since 2024; Section 10 does not claim otherwise.

Memory columns are gated rather than merely zero-valued. Zeroing the value is not enough, because the slots still take attention mass out of the softmax and starve the sequence: a zero value perturbs the output by 0.3, an additive logit bias of -10 on the memory columns by 2.6×10^{-5}. The gate opens to 0.34 once trained.

2.5 MemK, curation inside the memory

Insertion only ever grows the tree, and beam descent keeps two candidates per level, so every additional sibling is another chance for the correct branch to lose the cut. Retrieval therefore degrades as a function of the tree's shape rather than its content. MemK acts on two fronts. At write time a breadcrumb is treated as a suggestion: if the level it would join is already at the crowding limit, the fact is grafted under a better-separated ancestor. On tending, near-duplicate siblings merge and single-child interior nodes collapse, because a level where the beam never has to choose adds a hop and discriminates nothing.

The scorer is learned, six structural features per node, fitted against observed reachability rather than tuned thresholds. Content loss is checked explicitly: facts are counted before and after and the run fails if they differ. On the standalone

trees it took 186 nodes to 134 and 528 to 446 with recall held at 100% and zero facts lost. On a larger tree from a later run of the same curriculum it took 6,879 nodes to 4,013, with 3,637 facts before and 3,637 after and 2,866 nodes collapsed. The honest limit belongs in the same paragraph: descent visits moved by under one node and both arms saturate the probe at 100% recall, so this currently buys structure and not yet retrieval cost.

2.6 The specialist bank

Thirty-one specialists of two blocks each, consuming the same trunk output, never sharing gradients, trained one at a time with the trunk frozen. The bank is indexed along two axes over one taxonomy: nineteen behavioural shapes learned from transcripts, which is how to answer, and twelve topical branches learned from the knowledge corpus, which is what is true. Both are weighted simultaneously.

The argument for weighting both rather than choosing between them is structural. A top-1 gate has one slot to spend and must decide between answering in the memory-write shape and answering about Home. A dense composer does not face that decision. The same twelve-branch taxonomy is what the memory descends, what the knowledge corpus is sharded by, and what the bank is indexed by, so the three cannot drift apart.

XE4-S trained with no knowledge corpus, so its twelve topical specialists sat at initialisation throughout. That is a limitation and it also supplies a control that would have been expensive to construct: a specialist block is residual, so an untrained one is approximately the identity and emits the trunk's own prediction. Section 6.1 uses it as the floor, and Section 7.3 uses the composer's treatment of those twelve as evidence. Every knowledge-axis number in this paper comes from a companion run of the same architecture that did have a knowledge corpus, and is labelled as such.

2.7 The composer

Two linear maps and a softmax, 24,638 parameters, the smallest component in the system by an order of magnitude:

$$w = \mathrm{softmax}((W_{\mathrm{x}} \cdot \mathrm{LN}(h) + W_{\mathrm{m}} \cdot m + b_{\mathrm{use}}) / \tau) \quad \textbf{(2)}$$

The size is deliberate. The claim is not that the composer has capacity, it is that m is in its argument list. W_{m} is zero-initialised, so an untrained model is exactly a router and the memory term has to earn its contribution rather than arriving as noise. It did earn it: norm 11.22 against W_{x}'s 12.14, which is 48% of the routing signal.

Three tension dials default to previously measured behaviour. Descent tension is a margin rather than a temperature, because top-k is invariant to scale and dividing the scores changes nothing about which branches survive. Composition tension divides the composer logits, which makes the gate limit reachable as an ablation. Hop sharpness sets how decisively each hop contributes before the products compose.

The bias term b_{use} is written by use and never by gradient: one scalar per specialist, a buffer rather than a parameter, persisted in the state dict. The step is 0.0085 and the cap is 0.20, so one interaction moves a specialist's weight by under 0.2%, a hundred consistent ones move it about 4%, and then it stops. It leans the mixture toward specialists that have been earning their place. It is not meant to change which words come out, and Section 5.5 reports that it does not change anything else either. One level down, a per-node retrieval bias is written the same way and clamped to ±0.25 with decay, so popularity cannot outvote meaning.

2.8 Growth points

Adding a capability is five steps. Attach appends a specialist with its composer row zeroed and its bias at −12, which puts its mixture weight near 10^{-5}. Train carries gradient only in that specialist. Promote fits one composer row under a gradient mask that zeroes every other row. Gate evaluates against held-out data and writes only if nothing regressed. Keep leaves a standalone 14.2 MB file, so rollback is deleting a path.

Table 2: Every growth point is zero-initialised, and the claim that attaching a capability does not disturb the model is a measurement on a live system rather than a property of the initialiser taken on trust.

Operation	Measured
Attach a specialist	5.96e−07 output change
Deepen a specialist in place	0.000e+00 output change
Write a fact, gate at init	2.6e−05 output change
Write two facts	5 nodes, +1,920 params
Cost of one specialist	3.55 M params, 14.2 MB
Attach and swap time	0.074 s, 3.0 ms

Promotion is a distinct stage and the design fails without it. The attach leaves the specialist near 10^{-5} weight and the composer is frozen while that specialist trains, so with no promotion stage the specialist is trained, admitted, and contributes nothing. The gradient mask on promotion is not optional either: training the whole composer would let a new capability silently reweight every existing one, and the acceptance gate measures the model as a whole and would not see it happen.

2.9 The harness

The model emits text and the harness makes it an action. The harness is deliberately not trusted to the model. Schema validation refuses a call whose arguments do not match exactly rather than repairing it. Tool arguments are recovered from the utterance by pattern rather than by trusting what the model generated, after a defect in which the harness fell back to the model's invented arguments and produced confident writes of wrong values. Irreversible tools require confirmation, and a small fixed table answers identity and capability turns. A 130 M model should not be the last line of defence before an action with consequences.

2.10 What the stack does not contain

Stated here so it cannot drift from the claims. There is no top-k and no load-balancing term. There is no online gradient update: writing a fact adds parameters and those parameters are trained in a batched stage, never by an optimiser step

during a conversation. There is no segment loop for long-form output. Multi-turn conversation has never been run against a trained checkpoint. There is no mirrored return path, which the ordering search of Section 6.7 measured and rejected. Anchor-token masking, the −100 label trick that puts loss on answer tokens only, is implemented elsewhere in the codebase and is not used in this curriculum.

3 Training curriculum

Five stages on one RTX 3090, each of which changes a different part of the model and freezes what came before.

Table 3: The five stages as XE4-S ran them. Stage C's budget is the one number in this table that was never chosen on evidence: it has been between 1,200 and 3,000 steps since the first day of the project because that is what it was set to, and Section 6.6 measures what it should have been.

Stage	What trains	Budget
A	trunk, general English	2.11 B tokens, 200,000 steps
B	trunk, transcripts	4,000 steps
C	19 task specialists, trunk frozen	1,200 to 3,000 steps each
D	memory: 3,582 keys to 6,752 nodes, depth 4, branching 2.1	query projection fitted on 4,000 pairs, similarity 0.992
E	composer, task corpus	2,500 steps

Two properties of that schedule matter more than the step counts. The trunk is frozen from stage C onward, so a specialist trained in month three cannot renegotiate what the language layer learned in month one. And stage E is the only stage that trains anything shared across specialists, which is 2,500 steps against stage A's 200,000. The hardest discrete choice in the model receives about 1.25% of the training budget, and Sections 6.4 and 7.4 are both downstream of that.

Specialist sampling was got wrong twice before it was got right. Interleaved masking gives full input coverage and roughly 10% gradient density, which for XE4-S is 14.7 M effective tokens per specialist, 4.2 tokens per parameter against the trunk's 123. Contiguous slicing gives roughly 100% density and only ever shows a specialist hidden states produced by its own shape, which scored worse end to end, 36.5% against 57.9%. Hybrid sampling draws half of each batch from the target slice and half from the whole corpus, which keeps the input distribution and raises the effective token count by an order of magnitude.

Stage C reads the grouped corpus and stage E reads the interleaved one, and that is not a convenience. The composer's routing loss needs several shapes inside one window to discriminate between. A grouped corpus puts one shape per window and the stage collapses: loss 0.5153 interleaved against 10.08 grouped, with identical specialists.

4 Data

4.1 Three corpora

Three corpora with three jobs, because an earlier attempt interleaved general prose with task transcripts in one run and produced no measurable language ability. A mixed corpus is two objectives competing for one gradient rather than a language layer.

Table 4: The three corpora. XE4-S trained on the first two. The twelve knowledge branches were present in the bank and never trained, which Section 2.6 explains and Section 7.3 makes use of.

Corpus	Source	Tokens	Job
general	FineWeb-Edu [14]	2,107 M	how English sounds
task	our construction grammar	199 M	how to answer
knowledge	Wikipedia [15]	205 M	what is true

The general corpus is filtered only for what teaches something wrong. It is deliberately not passed through the construction-coverage selector, which keeps the 11% of sentences richest in rare constructions and strips exactly the ordinary declarative prose that makes a model sound normal. The knowledge corpus is routed to one of twelve branches before a token is written and then filtered to sentences that assert something: 40 to 320 characters, a relational verb from a closed list, and a named entity or a digit. Articles that do not route confidently are dropped rather than pooled, because a branch index becomes a specialist id and a junk-drawer specialist is worse than one specialist fewer. On a 400-article sample the routing rate is 42%, 167 of 400, with 43% of words kept.

4.2 The label channel

One uint8 per token, aligned element for element with the uint16 token stream, written at corpus build time. Labels 0 to 18 are behavioural and interleave through the transcripts; 19 to 30 are topical and are contiguous per branch. It is used twice, and using it twice is the point. Stage C masks the loss so that specialist *i* trains on exactly the tokens labelled *i*. Stage E uses the same bytes to supervise the composer, so the mixture is taught to weight the specialist that owns each token.

One artifact both partitions the training and defines the composition target, so no second annotation exists that could disagree with the first. Adding a capability means adding a label value and the data that carries it, with no architectural change. Corpus-label routing of this kind is not new: DEMix [4] assigned every token in a sequence to an expert by domain label, dismissed load balancing outright, and mixed densely over the whole bank at inference, in 2021. Section 10 keeps that in view. What is ours is the per-token granularity, the dual use of one channel as both partition and regression target, and the setting being assistant behaviour rather than document domain.

4.3 The tokenizer, and the brace it cannot write

A 16,384-entry byte-level BPE [16], fitted once on general English and thereafter loaded, never refitted.

It was fitted with an alphabet that contains no brace, and that is a property of this model rather than a bug that was fixed somewhere else. The general corpus filter drops any paragraph containing `{}[]<>|` as markup, so the tokenizer was fitted on 1.5 B words containing not one brace and the byte

never entered the alphabet. XE4-S learned to emit tool calls in the right shape and has no character with which to write them: every { encodes as <unk>. Every tool call this model emits is therefore malformed as text, and the harness's balanced-brace stop condition never fires, so generation runs to the token budget or to <|end|> instead of stopping when the call closes. A corpus filter decided the model's alphabet.

This is a real defect and it is not the reason for the accuracy gap in Section 5.1. The scorer has carried a brace-tolerant regular-expression fallback for the whole life of the project, applied to every model equally, so the missing brace was checked as a cause and rejected. It costs the model its stop condition and its well-formedness, not its tool choice.

4.4 Hard frames

Four syntactic frames are held out of the training distribution and appear in the probes: oblique, reported, cleft and negated. They are the frames a transfer-matrix sweep on this project measured as not transferring from any other construction family, so they have to be learned directly [20], and the corpus carries them at 0.09 to 0.30% of user turns. Searching token subsequences rather than decoding, 20,030 windows of 512 tokens, 5.15% of the corpus, hold at least one: oblique 7,260, reported 8,375, cleft 2,734, negated 1,931.

So the composer does meet them across batches. The dilution is inside the window, where about one turn in ten carries the frame and the routing loss spreads over all 512 positions.

5 Evaluation

5.1 Tool choice, end to end

178 held-out probes, tool-choice exact match, greedy decoding. The dense baseline `mono1` is a 17 M monolith trained on the same 700,000 sessions under an 8,192-entry BPE fitted on the task corpus, so the content is matched and the tokenizers are not.

Table 5: Standing accuracy on 178 probes, tool-choice exact match. Intervals are Wilson [19]. Making the memory path live moved the score by one probe, which is a result rather than a disappointment; Section 5.4 says why.

Model	Accuracy	Note
mono1, dense monolith, 17 M	82.6%	[76–87]
clm2, predecessor	60.1%	different tokenizer, memory inert
XE4-S, memory path off	57.9%	103/178
XE4-S, memory path live	57.3%	102/178, [50–64]
XE4-S, after the read fix	36.5%	Section 6.5

Composed against ablated validation loss at the end of the run is 0.3898 against 0.4055, which is the cleaner statement of what the memory contributes to the loss even though it moves the benchmark by one probe.

The deficit is one failure mode rather than a spread. XE4-S emits no call at all on 50 probes and the wrong tool on 25; the baseline emits no call on 0 and the wrong tool on 31. On wrong-tool confusions we are ahead of the dense baseline. The entire deficit is failing to act, and it is concentrated in the four held-out frames of Section 4.4.

Table 6: The gap, split by failure mode. Wrong-tool counts are quoted as 25 for the memory-off run and 26 for the memory-live run; they are different runs rather than a contradiction.

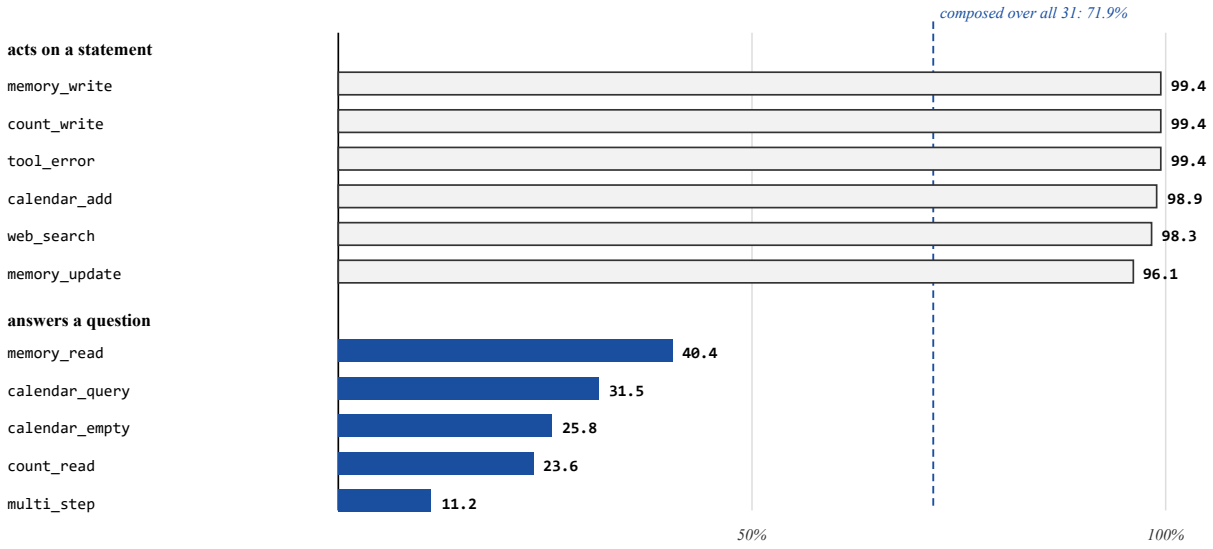
Model	No call	Wrong tool
XE4-S	50	25
mono1	0	31

Four causes were tested and rejected before the real one surfaced. Not the tokenizer, because the scorer's brace fallback applies to every model equally. Not the prompt format, because the corpus is <|user|> {text}\n followed directly by the call with no session prefix and the probe format matches exactly. Not the frozen trunk, because 178 of 178 probes have at least one specialist that opens, with a median of 12 of 31 opening individually. Not dilution by the abstaining tail, because sharpening the blend buys at most 2.3 points and truncating it is a separate question that Section 6.3 answers in the other direction.

What remains is where the composer puts its mass. On the 128 probes it gets right, 0.883 of the mixture sits on specialists that would open a call. On the 50 it fails, 0.285. Of those 50, seventeen put top weight on `blunt_answer`, which is a genuine selection error though a defensible reading of “I keep forgetting my tyre pressure again”, and fifteen put it on `memory_read`, which is the right shape and fails anyway. <|tool_call|> is rank 2 on eight of eight failures inspected, losing near ties such as 0.608 to 0.375.

5.2 The deficit is four specialists

Those fifteen right-shape failures are the thread. Figure 3 is the single-path opening rate on the same 178 probes, one specialist run alone.



Single path, one specialist run alone on the 178 held-out probes. The seven no-tool shapes sit at 0.0%, correctly, and are not plotted.

Figure 3: Every specialist whose shape acts on a statement opens a tool call on 96 to 99% of held-out phrasings. Every specialist whose shape answers a question manages 11 to 40%. The training corpora open at near-identical rates across that divide, so this is not a label-frequency artefact.

The training corpora open with a call at rates that are near-identical across the divide: `memory_read` 60.2% against `memory_write` 61.2%, `calendar_query` 62.0% against `calendar_add` 61.0%, `count_read` 54.9% against `count_write` 55.9%. The read specialists were not trained to abstain more often. They behave differently only on held-out phrasing.

The likely reason is the shape of the input rather than the frequency of the label. Read turns in training are predominantly questions, of the form “what is my bin day?”, while the probes are statements that imply a read, of the form “I keep forgetting my tyre pressure”. The write specialists are trained on statements and meet a statement; the read specialists are trained on questions and meet a statement.

And the corpus routes the hard frames straight at the specialists that abstain. `reported` is 100% `memory_read`; `oblique` is 37% `calendar_query` and 28% `memory_read`; `cleft` is 38% `memory_read`. Routing a probe correctly hands it to a specialist that emits `<|end|>` six times in ten, which is why 15 of the 50 failures route to the right shape and fail regardless.

5.3 Arguments, and a benchmark that measured a tokenizer

Every accuracy figure in this project before this one was tool-choice exact match. Nothing had ever scored the arguments. Two live outputs from the evaluated checkpoint make the gap concrete. “What's my bin day?” produces `memory_get` with the key `user.allotments`. A request for the opening hours of a leisure centre produces `web_search` with the query “the opening hours of the the stonework centre”. Right tool both times, useless both times, and counted as successes by every number the project had published.

Table 7: Four-level argument score on the probe set's invented entities. The last two columns are the ones nobody had ever looked at.

Model	Call	Tool	Keys	Values
XE4-S, 130.6 M	67.1%	50.0%	0.0%	0.0%
mono1, 17 M dense	100.0%	81.4%	81.4%	81.4%

That looked disqualifying. Four candidate causes were tested and killed first. Composition blurring the copy was killed, because single specialist paths also score 0.0% on values. The memory key-value path displacing the prompt was killed, because memory off scores 0.0% too. Stage C masking away the argument tokens was killed by reading the label channel, which gives the arguments the same shape label as the user turn. The model being unable to copy at all was killed by a teacher-forced probe, which copied `harbour survey` correctly, produced `Crocus Lane` for `Cragridge Lane`, correct first piece and then derailed, and substituted `the lake house` for `Quillwick Rise`.

The cause is subword length, and whose tokenizer had seen the probe generator.

Table 8: Subword length of the probe generator's invented proper nouns. The baseline's BPE was fitted on the task corpus, so the generator's inventions are single merges in it.

Entity	Ours, 16,384	mono1, 8,192
Quillwick Rise	5	2
Cragridge Lane	5	2
harbour survey	3	4

Our BPE was fitted on 2.11 B tokens of real English, which contains no “Quillwick”, so the inventions shatter into five rare pieces. Copying two tokens is a different task from reproducing five rare subwords in exact sequence, and `Cragridge` becoming `Crocus` is what derailing after the first piece looks like.

The control settles it. Re-run with entities a real user would say, the leisure centre, Manchester, council tax, bleeding a radiator:

Table 9: The same evaluation on real entities. Neither model reproduces an exact query at this scale.

Model	Tool choice	Query exact
XE4-S	62%	0%
mono1	100%	0%

The baseline also scores 0%. Its 81.4% was its tokenizer having memorised a synthetic generator rather than an ability of the model, and the argument gap was largely an artefact of the benchmark. On real text our tokenizer is the better of the two, 36 tokens against 44 for ten real entities, and worse only on the invented ones, 37 against 28.

Two things follow. The tokenizer should not be rebuilt to match the generator, because that trades real-world quality for a benchmark number. And the remaining real gap is tool choice on real phrasings, 62% against 100%, which is the one worth work. This is the most important methodological finding in the paper: a benchmark generated by the same machinery that produced one model's tokenizer measured that tokenizer.

5.4 Memory does not change behaviour

The tool benchmark cannot exercise memory. Across 60 probes the memory context has mean pairwise cosine 0.9883 and a deviation-to-mean ratio of 0.0955: on a single-turn probe with nothing personal stored, the descent lands in the same place regardless of what is asked, so roughly half the routing signal is a learned constant and ablating memory costs 1.2 points for that reason.

The memory-contingent benchmark exists to separate a model with state from a model without one. Two of its arms have the fact outside the 512-token window and one never stored it at all, so the visible context is identical. A model whose only state is its context must answer both alike, right on one and wrong on the other, which is 50% by construction at any size and any training budget. Widening the window moves the fact further back by a constant and does not change this.

Table 10: The memory-contingent arms, 60 probes each, memory path live. The finding is in the third column rather than in the totals.

Arm	Correct	What it said
in-window, control	58.3%	memory_get 35, no call 25
out-of-window	51.7%	memory_get 31, no call 18, web_search 7
no-fact	35.0%	memory_get 31, no call 21, web_search 6

Together the two measured arms score 43.3% against the 50% cap, a margin of −6.7 points. It does not clear the cap. `memory_get` is emitted 31 times in each arm, identical whether the fact was stored or was never stored at all. The fact is written, the tree is navigable, the key-value path is live, and none of it reaches the decision.

The in-window control at 58.3% is a failing control and is the reason −6.7 should not be quoted as precise. It is not a separate mystery either: every probe in every arm asks a read question, which routes to `memory_read`, which opened on 40.4% of held-out phrasings at the time they were scored. All three arms are measured through the component of Section 5.2. The differential survives that, because a uniformly depressed read path lowers both arms equally and cannot produce 31 against 31.

The likely cause of the equality is not a defect in the mechanism. Every `memory_read` session in the corpus has the fact present in context and the tool result hands the value back, so calling memory is never wrong and never informative. The model learned that a question about a personal fact calls `memory_get`, as a property of the phrasing. It was never given a case where the same phrasing needed a different answer depending on what was stored.

5.5 Growth by use moves nothing

The use loop is the architecture's central claim, that serving a user improves the model. Two arms run over an identical interaction stream under the same seed. The live arm writes any stated fact to memory and nudges the specialist that served each turn by whether it was right; the frozen arm does neither. There is no gradient, no optimiser and no training step in either.

The mechanism does run, and had never run before. b_{use} was 0 of 31 nonzero in every checkpoint in the project; after 20 interactions it was 17 of 31, and the memory grew from 6,752 to 6,753 nodes. Over 300 interactions, held-out accuracy in both arms sat at 0.5167 at every checkpoint measured, at 0, 75, 150, 225 and 300. The live arm moved +0.0 against the frozen one.

That is a real null on the thing the architecture is for. It is also the one test a dense transformer cannot produce a difference on, since its behaviour on turn one and turn ten thousand is identical by construction, so the test is worth keeping and re-running against a model whose read path works.

5.6 Degeneration and throughput

Generated past anything the corpus contains, output novelty collapses to 0.00 by token 144, measured as the fraction of recent tokens not already seen in a 48-token window. Template answers are short and repetitive, so repetition is what the loss rewards.

Throughput is 4.6 tok/s at batch 1, 7.2 at batch 4 and 7.7 at batch 8, with peak VRAM of 537 to 555 MB. A separate measurement on the acceptance prompt mix gives 4.2 tok/s; both are real and they differ by prompt. The batched specialist bank runs 11.5× faster at $k = 31$ and agrees with the sequential loop to 1.55×10^{-6}, and it is imported by nothing. This failure has a verified fix sitting unused.

5.7 The acceptance test

Seven owner-written criteria rather than a benchmark, run against the checkpoint this paper describes. The result is 3 of 7.

Table 11: The acceptance test, first run, on `E_composer.pt`. Three of the four failures have identified fixes that are either unbuilt or untrained.

Check	Result
English reply	pass
Right tool for a calendar add	fail
A paragraph, not a fragment	fail
Search, with a query from the request	pass
Faster than 20 tok/s	fail, 4.2
The model changed through use	pass
Reached for memory past the window	fail

What passed is worth quoting rather than scoring. Asked how a circuit breaker trips, the assembled system replied, unedited:

> Two mechanisms. A bimetallic strip bends as it heats and trips on sustained overload; an electromagnet trips near-instantly on a short. That's why a slow overload takes seconds and a dead short takes milliseconds.

Accurate, fluent and correctly structured, from 130.6 M parameters trained from scratch on one consumer GPU with no pretrained base. Every remaining failure in this paper sits above the language layer. Search ran end to end, with the wrong entity in the query. The use loop moved memory from 6,752 to 6,754 nodes and use-bias from 0 to 3 of 31 specialists with no gradient and no retraining. Two of the failing checks produced identical wrong text from two unrelated prompts, which is the model falling into the same attractor rather than making random errors. Check seven failed cleanly: with the name 900 tokens outside a 512-token window, the model hallucinated a `calendar_lookup` rather than consulting memory.

6 Ablations

6.1 Composition against single paths

On the opening decision over the 178 probes, composed over all 31 opens a tool call 71.9% of the time, the mean trained single path 43.1%, and an untrained specialist 36.9%. An untrained specialist is a residual block at initialisation and therefore approximately the trunk's own prediction, which makes the last row a trunk control rather than a weak expert. Composition is worth +28.8 points over the mean single path and about +35 over the trunk.

Restricted to trained specialists only, 178 of 178 probes still have one that opens, against an untrained-specialist floor of 148 of 178. The headroom is real and the floor is high, so that figure says the ability exists in the bank rather than that it is rare.

Where routing is correct, composition also beats the best single specialist. On Home, the one knowledge branch the composer of the companion run could route, composed scores 5.0303 against best-single 5.1764, a gain of +0.1461 nats. A top-1 gate returns exactly the best single expert by construction, so that margin is the part a gate cannot reach, and it appears wherever routing is correct and nowhere it is not.

6.2 Are the specialists differentiated

Twelve knowledge branches in the companion run, single path with the specialist index forced so that the trunk and memory are held fixed, 24 windows of 512 tokens per cell. Own-branch mean loss is 5.4786 against 5.8999 on every other branch, a gap of +0.4212 nats, and 12 of 12 branches are best served by their own specialist. The bank is not thirty-one copies of one function.

On the task axis the recorded figures are a +1.81 nat own-against-other gap and composition beating the best single specialist on all 19 domains at a mean +0.91 nats. Those two numbers appear only in a summary, with no detail table or cited artifact behind them anywhere, unlike the knowledge-axis figures. They are reported here with that caveat attached and should be re-derived before they are leaned on.

6.3 Truncation is free, which retracts a claim of ours

This entry was previously reported the other way and the correction matters more than the number.

Serving only the smallest set of specialists covering top_p of the mixture was measured at a cost of 11.8 points at $p = 0.99$ and 21.3 at 0.90, and that figure went into a paper as the sharpest evidence that dense composition beats top-k. It was our own bug. The kept set was selected by cumulative sum over each specialist's maximum weight across positions, which is not a distribution and sums to roughly 9 over 31 experts, so the threshold was crossed after about three. The measurement described truncation to two or three specialists while the parameter said seven or more.

Table 12: Truncation, with the selection rule corrected to sum over the mean across positions, which does sum to one. Call recall on the opening decision, 178 probes.

top_p	Specialists kept	Call recall
1.00	31	71.9%
0.99	15	71.9%
0.95	10	72.5%
0.90	8	73.0%
0.80	5	72.5%
0.60	3	65.7%

Truncating to eight of thirty-one is marginally better than the full bank, not 21 points worse. The claim that dense composition beats top-k is withdrawn. What replaces it is smaller and true: serving can truncate to eight with no loss, which prices the 3.7× speed-up from truncation, 22.0 to 79.3 tok/s on GPU, at zero accuracy.

The defect was found because an adversarial prior-art search surfaced Chen and Yao [7], who argue that most reported top-k truncation cost is a renormalisation artefact. The search was looking for papers that would embarrass us and found our own bug instead.

6.4 Training the composer harder does not help

Most of one long day was spent assuming the composer was at fault. Three independent attempts to improve routing each raised training routing accuracy and each lowered the

benchmark. Longer plain training went 71.9 to 73.6 to 70.2 with no trend, while training routing accuracy rose 0.760 to 0.795. Frame-weighted sampling, drawing half of each batch from the hard frames, went 71.9 to 69.1 to 69.1 to 67.4 to 66.9, monotonically down, while training accuracy rose 0.760 to 0.791. A hidden path with real capacity took training accuracy 0.760 to 0.863 and held-out opening 71.9% to 62.9%, the worst of the three.

A fix that reliably makes things worse is a fix to something that was not broken. The hidden path remains in the code defaulting to absent, because a negative result that is reachable is worth more than one that has to be taken on trust, and because attaching it is exactly the identity, $0.000\times10^{+00}$ against model run-to-run noise of 1.9×10^{-3}.

6.5 Weighting the decision position

Ordinary language-model loss gives the one position where a specialist decides whether to act the same weight as the other 511. Weighting that position 8× for 1,500 steps per specialist at batch 12, with the trunk, the composer and the other twenty-seven specialists frozen, took the four read specialists from 23.6–40.4% opening to 84.3–98.3% in 10.7 minutes of training. `memory_read` crossed 90% within 300 steps and 31 seconds.

End to end that is a regression from 57.3% to 36.5%. No-call misses fell from 50 to 31 and wrong-tool misses rose from 26 to 82, with `memory_set` emitted where `memory_get` was wanted rising from 14 to 59. The specialists learned to open without learning what to open with, because the weight sat on the opening token and left the tool-name token immediately after it at 1. Silent abstention was converted into confident wrong answers, which on this scorer is worse.

Carrying the weight twelve tokens further fixes opening for good, 76 to 99% across all four, and leaves right-tool rate at 18–43% and oscillating rather than climbing. `memory_read` swings 50 points between checkpoints 300 steps apart. A module that unstable has not converged on the decision.

6.6 How much specialist training is enough

Volume was measured directly rather than assumed, at eight times the stage C allocation on one read specialist, with right-tool rate scored every 3,000 steps. Right-tool rate goes 28.3% at stage C only, 53.3% after 3,000 additional steps, and then oscillates between 35.0% and 53.3% for the next 21,000 while training loss moves 0.1954 to 0.1831. Opening rate over the same span goes 79.8% to 91.0% and back to 74.2%.

Doubling stage C is worth about +25 points on right-tool rate and everything past that is noise. The specialists were undertrained by roughly 2×, not by the 10× the token-per-parameter ratio implies. And volume is not the reason held-out phrasing fails: a module that has converged on its own distribution and still cannot handle unseen phrasing is reporting that the distribution lacks the phrasing, not that it needs more passes over it. This sweep was run on a later checkpoint of the same specialist geometry rather than on the checkpoint evaluated here, which Section 9 records.

6.7 Layer ordering, and what happened when it was replicated

The layer ordering was not chosen. A parameter-matched search over all six orderings of a shared language layer, a routed specialist and a shared memory ran eleven arms identical in blocks, parameters and inference depth, each against a monolith sized to the system total. Language first, specialists next, memory last scored 0.7877 against the monolith's 0.7350. Every specialist-first arm collapsed to 0.6245 through 0.6514, below the monolith in all four cases. Mirroring the winning order cost 4.9 points. That was the strongest positive result the project held, and it predates the XE4 line.

An adversarial prior-art search warned that the field treats gaps of this size as inside seed variance, and that seed replication would be worth more than any citation. The replication was run and it does not replicate.

Table 13: Four seeds per arm, identical data, only initialisation and batch order differing. The original +5.27 was the best draw of four and should be quoted as a mean and a spread or not at all.

Seed	System	Matched monolith	Margin
42	0.7877	0.7350	+5.27
101	0.7536	0.8285	−7.48
202	0.8145	0.7828	+3.18
303	0.7405	0.7875	−4.70
mean			−0.93

The mean over four seeds is −0.93 points, the spread is 12.8 points, the sign flips twice and two of four arms win. The mirror penalty behaves the same way: computed from the same four arms it is +4.9, −1.0, +2.6 and +0.6 points, a mean of +1.8 with one seed where the mirror is better. A compositional system does not beat a parameter-matched monolith here, and we do not claim it does. Cortes and colleagues [23] and Li and colleagues [24] have published in that neighbourhood at far larger scale, and Section 10 defers to them.

What survives is weaker and still useful. The eleven arms produce a large spread from ordering alone, and the specialist-first collapse to 0.62 through 0.65 is far outside this seed variance and remains the reason the language layer goes first.

6.8 Sampler and corpus layout

Stage E's knowledge stream was measured directly against the real corpus over the exact 1,250 steps at batch 24 that the stage uses, counting the label channel. A sequential cursor shows 1 of 12 labels with Home at 100.00%. Random block draws show 12 of 12 with Home at 9.00%, and after the fix every branch appears in proportion to its size, from Person at 3.10% to Nature at 9.42%. Section 7.4 is what that cost.

7 Routing at $N = 31$, against a collapse at $N = 3$

This is the result the paper is built around, because it is the only comparison available where both sides were measured by the same lab on its own record.

7.1 What NCN was, and how it failed

The previous architecture composed NCN Module Specialists: narrow parameter-efficient delta weights over frozen pretrained bases of one to three billion parameters, rank-16 LoRA at about 16.7 MiB and rank-8 at about 8.4 MiB, packaged as `.ncn` artifacts. Each carried a capability capsule, an explicit scope and exclusion list, a bound validation set and a provenance manifest that the router consumed at inference. Selection was closed form: a pooled query embedding scored against the per-specialist capsules by cosine similarity or by GSVD-routed capsule similarity, producing a sparse active set. The ship gate was 85% top-1 route accuracy [1].

At $N = 2$ it is clean, 18 of 20 combined across bases, with both misroutes low-margin general-to-architecture confusions on the cross-base wrapper. At three and above, over capsules from similar domains, it collapses. Figure 2(a) is the measured saturation curve on a 30-prompt held-out set.

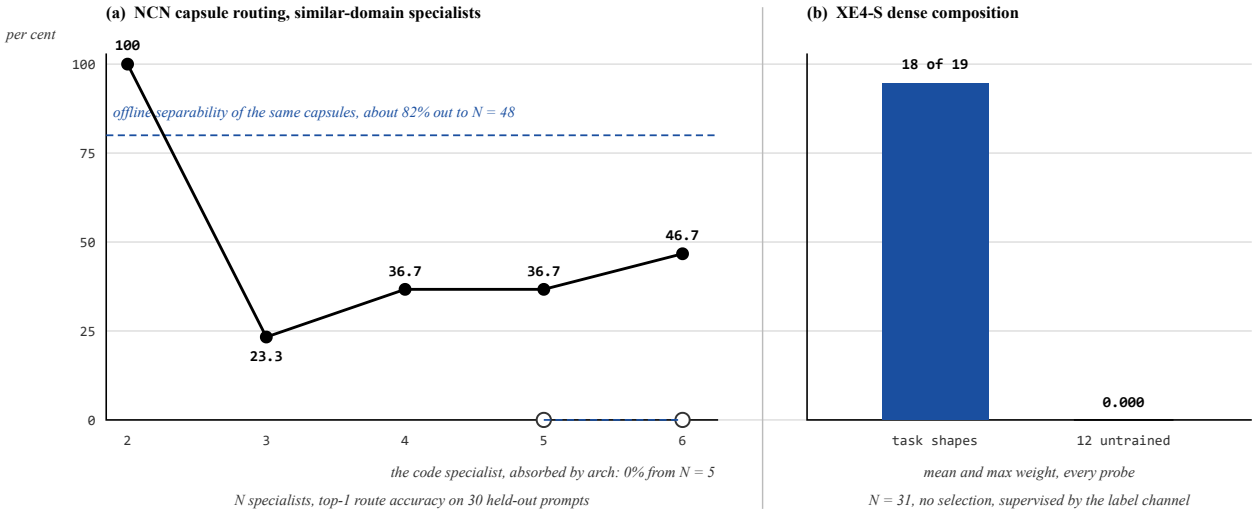


Figure 2: The routing comparison. (a) NCN capsule routing saturating as the active set grows, with the `code` specialist absorbed by `arch` and pinned at 0% from $N = 5$. The dashed rule is what the same cosine signal achieves offline on the same capsules, about 82% out to $N = 48$ against a random floor that falls from 12.5% to 2.1%. The roughly 30-point gap between that ceiling and deployment routing near 47% is the finding: the geometry holds and the deployment projection does not. (b) XE4-S at $N = 31$, where nothing is selected. The metrics are different quantities and the caption in the figure says so.

Three follow-up measurements sharpen it rather than soften it. Mean-centring the capsules tripled decision margins and left accuracy essentially flat, with $N = 6$ rising only from 46.7% to 50.0%, so the failure is a query-to-capsule representation mismatch rather than a geometry problem. Expanding one specialist's corpus from 237 to 4,361 examples made $N = 6$ worse, 53.3% down to 46.7%, and collided the `diag` and `dialog` capsule cosine to 0.926, because a single mean-vector capsule becomes more diffuse rather than more discriminative as its corpus broadens. And a learned replacement, the NCN v2 router, did worse than closed form: every one of 30 prompts across all six specialists routed to a single node, mutual information exactly 0.0 bits. It was abandoned.

The space itself is separable. Offline, the same cosine signal separates about 82% of targets up to $N = 48$, with GSVD tying plain cosine-to-mean, against a random floor falling from 12.5% to 2.1%, a ten-seed stability of 0.00 and mutual information reaching about 88% of maximum at $N = 48$. The proposed remedy, a hierarchical gate that descends a topology region before selecting inside it, is specified and untested, and is that program's fulcrum experiment.

7.2 What XE4-S does instead

Two mechanism differences, and both are load bearing.

NCN had to choose. Selection by capsule similarity produces a sparse active set, so as N grows the router is solving a harder and harder discrimination over a representation that was never trained for it, and a single dominant capsule can absorb everything around it. XE4-S composes all thirty-one specialists with non-zero weight and selects nothing, so there is no top-k, no active set, and no load-balancing loss anywhere in this work.

NCN derived its routing signal from capsule similarity, which is a property of the specialist's own corpus. XE4-S takes it from a supervised corpus-side label channel, one uint8 per token, written at corpus build time. The same bytes that mask the loss for specialist i in stage C supervise the composer in stage E. Nothing learns to route from embedding geometry, because the assignment was fixed in the data.

7.3 The measured outcome at $N = 31$

The composer routes 18 of 19 task shapes correctly in distribution. The bank it does that in is thirty-one wide, and the twelve knowledge specialists in it were never trained, because XE4-S ran with no knowledge corpus. The composer gives those twelve exactly 0.000 weight, mean and max, on every probe, with no mechanism asking it to. There is no dead attractor and nothing absorbing the mass of a neighbour.

7.4 Where XE4-S reproduced the collapse, and why

The knowledge axis collapsed in exactly the NCN v2 pattern, and closing that loop is the reason this section can be trusted.

In the companion run that did have a knowledge corpus, the composer routed 12 of 12 knowledge branches to `know:Home`. One destination, every input, which is the zero-bit failure in a different notation.

The cause was identified by arithmetic rather than inferred. The batch sampler walked a sequential cursor from block zero. Stage E runs 2,500 steps, half of them on knowledge at batch 24, so its knowledge stream consumes 30,000 blocks of 512 tokens, 15.4 M tokens, starting at block zero, and the knowledge corpus is laid out branch by branch with Home occupying blocks 0 through 34,921. The stream never left Home. Eleven of twelve labels were never once shown. Section 6.8 has the direct measurement.

So the collapse here was data exposure rather than routing machinery, and that is the difference that matters against the NCN case. There the mechanism was varied four ways, cosine-to-mean, GSVD, a learned router and shrinkage-LDA, without isolating a cause, and the remedy remains a specification.

The repair is real and partial, and overstating it would undo the point. After the sampler fix, branches routed to their own specialist went from 1 of 12 to 3 of 12, mean mass on own specialist 0.111 against a uniform 0.032, and the mean cost of composing against the best single specialist improved from −0.3093 to −0.1923 nats. The collapse moved from `know:Home` to `know:Society` rather than dispersing. Stage E gives the knowledge half 15.4 M tokens to learn a twelve-way split on top of a nineteen-way one, and that remains unresolved.

One more parallel is worth recording because it cuts against both architectures. In the NCN line the obvious remedy made things worse: a corpus nearly twenty times larger dropped $N = 6$ accuracy. In the XE4-S line the obvious remedy made things worse too, and Section 6.4 is the record of it. Neither system's routing problem was ever short of the thing the intuition said it was short of.

7.5 What the comparison does not establish

The two systems are not a controlled head-to-head. NCN routed LoRA deltas over frozen pretrained bases, selected by geometry, scored as top-1 route accuracy on 30 held-out prompts over two to six similar-domain specialists. XE4-S trains a bank from scratch at thirty-one and its routing is supervised. The metrics are different quantities.

What the comparison establishes is narrower than a benchmark win and more useful than one. The collapse that stopped the previous architecture does not appear here, and the two mechanism changes of Section 7.2 are the candidate reasons. It does not establish which of the two is responsible, and the experiment that would, supervised routing against geometric selection on one bank, has not been run.

Nor does it rescue the end-to-end model. XE4-S still loses to a dense baseline on tool choice, 62% against 100% on real phrasings. The 18 of 19 figure is in-distribution routing, measured on the distribution the composer was trained on, and out-of-distribution syntactic frames are where the assembled model fails. Dense composition also does not beat top-k at serving time, as Section 6.3 records. The claim is about the training and routing regime, that no selection has to be learned and no selection collapses as N grows, and not about needing all thirty-one resident to answer a turn.

8 Four bugs of one class

The instances matter less than the class, and this class produced four bugs in this system. State that is not a fixed-shape buffer, silently not kept in step with what it indexes. Loading a checkpoint without strict key matching drops an entry that has no destination in a freshly built model and says nothing; Python-side collections never enter the state dict at all. Nothing errors. The numbers quietly mean something else.

The memory tree, which is plain Python, never entered the state dict, and the memory context short-circuits to null when the node list is empty, which quietly demotes the composer to a plain router. That one was found and fixed months before the others, after which the search stopped.

The per-node value vectors and the latent slots on load. The values are a parameter list that a fresh model creates empty, and the latent slots are a buffer created at zero rows, so a checkpoint's 6,752 value entries had no destination and every one was discarded. The key-value accessor then returned null on the first line of its body because the value list was empty. The loader compounded it by restoring the tree after the state dict rather than before. The consequence is that the memory-as-persistent-key-value path was inactive in every evaluation this project has ever run, including the 57.9%, and everything stage D trained on top of it was discarded on every save and reload. After the fix a reloaded checkpoint reports 6,752 nodes, 6,752 values and a latent block of 6,752 rows, with the accessor active at slot norm 32.24.

The same tensors on prune. The curation layer's rebuild reindexed nodes, depths, leaf flags, keys, children and roots after a prune and did not reindex values, latents or affinities, all three of which are indexed by node. After any prune, node i carried whatever node i used to be, so its learned value belonged to a different fact. Curation was corrupting the memory it exists to protect, and the facts-lost count of zero was measuring the symbolic tree only. Verified after the fix on a 19-node tree pruned to 13: nodes, values, latents and keys all 13, each survivor keeping its own latent slot.

Checkpoints carrying no record of context length or positional scheme. The positional flag is a plain boolean and the context length is only implied by a tensor shape, so loading a checkpoint through a loader hardcoded to the other setting drops the positional table on shape mismatch and would have scored a model nobody trained. Fixed by writing a config block on save and inferring both on load.

Two adjacent classes produced the rest. A sampler assumption that holds for interleaved data and fails for grouped data appeared twice, as the knowledge cursor of Section 7.4 and as stage E collapsing on the grouped corpus, 10.08 against 0.5153 with identical specialists. Those are the same

assumption seen from two sides. And a filter upstream silently deciding something downstream produced the tokenizer alphabet of Section 4.3.

The guardrails are in place for each. The loader prints the value count beside the node count and flags a mismatch. The batch sampler takes a randomisation argument. The byte alphabet is seeded explicitly. The truncation selection rule sums over the mean.

9 Limitations

Every accuracy figure in this paper other than the argument scores of Section 5.3 is tool-choice exact match. Models were compared like for like, so the gap to the dense baseline is real, but the absolute numbers overstate how usable the model is by an amount nobody has established.

The context window is 512 tokens and cannot be widened without retraining, for the reason Section 2.3 gives. Multi-turn conversation has never been run against a trained checkpoint. Serving is 4.6 tok/s.

Two figures have weaker provenance than the rest and are flagged rather than dropped. The task-axis specialisation numbers, +1.81 nats and +0.91 nats, appear only in a summary with no detail table behind them. The seed sweep of Section 6.7 is four single runs per arm, which is what it took to destabilise the original claim and is not enough to establish a replacement.

Two further measurements come from neighbouring runs rather than from the evaluated checkpoint, and are labelled where they appear. Every knowledge-axis number comes from a companion run with a knowledge corpus, because XE4-S had none. The volume sweep of Section 6.6 was run on a later checkpoint of the same specialist geometry.

The memory-contingent margin of −6.7 points is measured through a read path that opens on 40.4% of held-out phrasings, and its in-window control fails at 58.3%. That benchmark has to be rescored after the read path works before its number means anything. The differential inside it, 31 calls against 31, does not depend on the totals.

The curation result buys structure and not yet retrieval cost: descent visits moved by under one node and both arms saturate the probe at 100% recall. A harder probe is needed before a retrieval claim is made.

10 What this paper does not claim

The architecture has not been shown to fail. The memory mechanism is built and separately verified, the growth points are measurably safe, and the specialists are measurably differentiated. What is missing is training that makes using any of it necessary, and the evidence for that is the same measurement in three places: behaviour identical across memory states, a use loop that moves nothing, and arguments that are never copied because the corpus never required copying.

The bare contrast between $w = \text{gate}(x)$ and Equation 1 is not a novelty statement. Retrieval-conditioned routing exists in kNN-MoE [8] and in RMS-MoE [9]. What survives is that the retrieved object here is runtime-written user content rather than a cached routing decision, and that the output is dense rather than sparse.

Corpus-label routing with dense mixing and no balancing loss is not ours either. DEMix [4] did all three in one 2021 paper, and SMEAR [5] adds a tag-routing baseline that deliberately omits balancing losses. Dense mixtures beating sparse ones is established across at least three papers [5, 6, 7], and our own measurement of it turned out to be a bug (Section 6.3). Whether anything can retract one named memory has a two-year-old answer in Larimar [10], which does it gradient-free with derived keys by flipping a sign. Admitting a new expert into a trained bank by training only the router is a promotion stage in everything but name and is published [12, 13]; what is ours there is the 10^{-5} measurement that says what happens without one.

The claim that a modular system beats a parameter-matched monolith is withdrawn at the scale we measured it, for the reason Section 6.7 gives, and the published statements of that direction [23, 24, 25] are at a thousand times the scale.

The dense-baseline comparison is not a verdict on compositional modelling. The benchmark it runs on determines the required shape from the input alone, so consulting memory buys nothing there and a router would score identically. The task that separates the two is memory-contingent, and on the version of it that exists the read path all three arms are measured through opens a call on 40.4% of held-out phrasings.

The routing comparison of Section 7 is not a controlled head-to-head, for the reasons Section 7.5 gives. It establishes that the failure which stopped one architecture does not appear in the other, and it names two mechanism differences as the candidate reasons. Both sides of it were measured by this lab, on its own record, with the failing side documented before the succeeding one was built. That is a narrower property than a benchmark win and it has one advantage over every other claim in this section: no prior-art clearance is required to state it, and nothing published later can retroactively occupy it.

11 What the failures imply for the next model

One diagnosis covers almost all of it. The corpus is template-generated, the specialists fit it closely, and they therefore learned the templates rather than the behaviours. Four of the five remaining failures are things the training data never required.

Copying. Every slot has a typical filler, so learning the filler scores as well as copying the entity. Measured at 0% exact argument match, on ours and on the dense baseline.

Consulting. Every read session sets the fact earlier in the same session and the tool result hands the value back, so calling memory is never wrong and never informative. Measured as `memory_get` emitted 31 times whether the fact was stored or never stored.

Hard frames. Oblique, reported, cleft and negated sit at 0.09 to 0.30% of user turns, and they are among the families that do

not transfer from any other construction and therefore have to be learned directly.

Saying more. Template answers are short and repetitive, so repetition is what the loss rewards. Measured as novelty collapsing to 0.00 by token 144.

The fifth failure, 4.6 tok/s, is not a data problem. It is a verified 11.5× speed-up that nothing imports.

What follows, in the order the evidence supports. Rebuild the corpus so that the easy strategy fails: not more data, but data where copying the typical filler, reflexively calling memory and repeating a template all score badly. The shapes for this are written and their label ids become new specialists, which is the growth mechanism of Section 2.8 doing what it was built for, and the trunk does not need retraining, so it costs stages B and C only. Set stage C to 6,000 steps, because doubling it is worth about +25 points on right-tool rate and the current 1,200 to 3,000 was never anything but a default. Import the batched bank, whose measurement exists and whose code exists and which nothing calls. Score arguments as a standing metric, because the distance between the right tool and an answer that works was invisible for the life of the project. Give the knowledge axis a stage E budget that matches the decision it makes, since it is the one place the collapse of Section 7 has not been cleared. And replicate before claiming; the ordering result is the cautionary case, four seeds and a mean of −0.93 quoted for months as +5.3.

One structural item belongs on that list and is not a corpus fix. The ordering search of Section 6.7 ran three functional layer types, and the one it called memory was a shared stack of blocks that every corpus passes through, holding what spans specialists rather than what belongs to any one of them. XE4-S carried the ordering forward and did not carry that layer. After composition it goes straight to the retrieval stage and the head, so thirty-one hidden states are summed and decoded with nothing shared in between to reconcile them. The measured consequence is exactly what that predicts: composition adds hidden states and then applies one head, so a weighted sum of twelve vectors meaning “call a tool” and nineteen meaning “end the turn” is a point between them rather than a vote, and nothing makes a majority win. Blending after the head instead, scored as a rescore with the same weights and never as a trained configuration, recovers 8 of the 50 failures and loses 2 of the 128 successes, a net +3.4 points. Attaching such a stack to a trained checkpoint is verified to be exactly the identity at $0.000\times10^{+00}$, so it can be added under the same growth discipline as everything else. It has never been trained here, and nothing in this paper depends on it.

12 Reproduction

The measurements in this paper come from scripts in the project directory, each of which answers one question. `compare.py` produces the headline accuracy figures and the memory ablation. `specialisation.py` produces the specialist-by-domain matrix and what composing is worth. `ceiling.py` produces the opening-rate ceiling. `read_fix.py --measure-only` produces the abstention table of Figure 3, and without the flag it applies and instruments the fix. `tension_sweep.py` produces the truncation and temperature sweeps. `contingent.py --make` builds the memory-contingent probes and `contingent_eval.py` scores them under separate model states per arm. `arg_eval.py` produces the four-level argument score. `memk_eval.py` produces the curation numbers with a fact count that fails the run if content was lost. `batched.py --verify --bench` produces the 11.5× and the 1.55×10^{-6}. `use_loop.py` runs the growth-by-use arms. `frame_index.py` produces the 5.15% frame density. `spec_audit.py` instantiates the model as training configures it and prints what is present and what is not built in one pass. The ordering arms and the seed sweep live in `compositional/rings.py`.

A note on artifacts rather than papers. Of the systems in this neighbourhood that actually train a model rather than wrapping a commercial API, exactly one published weights. DEMix [4] did dense composition over a specialist bank at 1.3 B in 2021 and let the weights evaporate; the checkpoint request on its repository was closed without a reply. Titans [11] has never released a line, twenty months after promising code, and every set of weights carrying its name is an adapter on somebody else's base model. Larimar's selective-forgetting result, the thing that makes the paper interesting, has no code in the repository the paper points at. A working artifact at any scale is a real and scarce contribution on the only axis that is checkable, and it is also the reason most of the comparisons we want cannot be run. Absence of an artifact is weak evidence of difficulty and no evidence of importance.

Prior work by the same author

Garren, T. J. (2026). *Towards Compositional Language Models: An Auditable, Falsification-Driven Program for Structured, Modular, Time-Aware Generation.* SOPHIA XT LLC, Version 2.0. The NCN Module Specialist line, its capsule routing and the saturation curve reported here are from that program. Both sides of the comparison in Section 7 are therefore the same author's work: the architecture that collapsed was specified, measured and published as a failure before the one that does not was built.

References

[1] Garren, T. J. Towards compositional language models: an auditable, falsification-driven program for structured, modular, time-aware generation. SOPHIA XT LLC, Version 2.0, 2026.

[2] Shazeer, N., Mirhoseini, A., Maziarz, K., et al. Outrageously large neural networks: the sparsely-gated mixture-of-experts layer. *ICLR*, 2017.

[3] Fedus, W., Zoph, B., Shazeer, N. Switch Transformers: scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 23:1–39, 2022.

[4] Gururangan, S., Lewis, M., Holtzman, A., Smith, N. A., Zettlemoyer, L. DEMix layers: disentangling domains for modular language modeling. *NAACL*, 2022. *arXiv:2108.05036*.

[5] Muqeeth, M., Liu, H., Raffel, C. Soft merging of experts with adaptive routing. *TMLR*, 2024. *arXiv:2306.03745*.

[6] Puigcerver, J., Riquelme, C., Mustafa, B., Houlsby, N. From sparse to soft mixtures of experts. *ICLR*, 2024. *arXiv:2308.00951*.

[7] Chen, X., Yao, H. Training-free halving of activated experts in fine-grained mixture-of-experts models. *arXiv:2609.04575*, 2026.

[8] Lyu, B., Murakami, S., Kamigaito, H., Zhang, P. Routing by analogy: kNN-augmented expert assignment for mixture-of-experts. *arXiv:2601.02144*, 2026.

[9] Rethinking MoE with retrieval-memory synergy: towards efficient expert coordination. *Proceedings of the ACM Web Conference*, 2026. DOI 10.1145/3774904.3792922.

[10] Das, P., Chaudhury, S., Nelson, E., Melnyk, I., et al. Larimar: large language models with episodic memory control. *ICML*, 2024. *arXiv:2403.11901*.

[11] Behrouz, A., Zhong, P., Mirrokni, V. Titans: learning to memorize at test time. *arXiv:2501.00663*, 2025.

[12] Morrison, J., Adhikesaven, S., Bhagia, A., Zaharia, M., Smith, N. A., Min, S. Train separately, merge together: modular post-training with mixture-of-experts. *arXiv:2604.18473*, 2026.

[13] MoE-LPR: multilingual expansion of large language models via mixture-of-experts with language priors routing. *AAAI*, 2025. *arXiv:2408.11396*.

[14] Penedo, G., Kydlíček, H., Ben Allal, L., et al. The FineWeb datasets: decanting the web for the finest text data at scale. *NeurIPS Datasets and Benchmarks*, 2024.

[15] Wikimedia Foundation. Wikipedia database dumps, English, 1 November 2023. CC BY-SA 4.0.

[16] Sennrich, R., Haddow, B., Birch, A. Neural machine translation of rare words with subword units. *ACL*, 2016.

[17] Xiong, R., Yang, Y., He, D., et al. On layer normalization in the transformer architecture. *ICML*, 2020.

[18] Vaswani, A., Shazeer, N., Parmar, N., et al. Attention is all you need. *NeurIPS*, 2017.

[19] Wilson, E. B. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22:209–212, 1927.

[20] Goldberg, A. E. *Constructions: A Construction Grammar Approach to Argument Structure.* University of Chicago Press, 1995.

[21] Press, O., Wolf, L. Using the output embedding to improve language models. *EACL*, 2017.

[22] Hendrycks, D., Gimpel, K. Gaussian error linear units (GELUs). *arXiv:1606.08415*, 2016.

[23] Cortes, C., Mohri, M., Zhong, Y. A theoretical framework for modular learning of robust generative models. *arXiv:2602.17554*, 2026.

[24] Li, et al. Can mixture-of-experts surpass dense LLMs under strictly equal resources? *arXiv:2506.12119*, 2025.

[25] Berges, V.-P., Oguz, B., Haziza, D., Yih, W., Zettlemoyer, L., Ghosh, G. Memory layers at scale. *ICML*, 2025. *arXiv:2412.09764*.