XE5: Compositional Language Modelling with Memory-Conditioned Specialist Mixing

Thomas Garren **Sophia**

SOPHIA XT LLC

sophiaxt.com

ABSTRACT

XE5 is a 127.5 M-parameter language model trained from scratch in which specialist sub-networks are *combined* by a learned composer rather than *selected* by a gate, with the combination weights conditioned on where a beam descent through a hierarchical memory of runtime-written facts terminates. A mixture of experts computes $w = \text{gate}(x)$; XE5 computes $w = \text{compose}(x, m)$. Because m depends on stored state the input does not determine, fixing the input does not fix the weights.

This report describes the system layer by layer. Three corpora with three separate jobs. A 16,384-entry byte-level tokenizer whose byte alphabet is seeded explicitly. One uint8 label per token that both partitions specialist training and supplies the composition target. A six-block trunk with rotary positions and a verified key/value cache. A memory read three ways: as a context vector summarising a descent, as persistent keys and values that every block attends over, and as a latent slot written by one projection and rewritable at its address. A learned curation layer inside that memory. A bank of 31 two-block specialists that runs as batched matmuls rather than a Python loop. A composer under 0.03 M parameters whose only distinction is that m is in its argument list.

Measurements accompany each layer. Rotary positions let a 128-window model run 300 tokens and agree with the cache to 1.6×10^{-7}; the cache holds decode throughput flat at 47.2 tok/s over 241 tokens of context where the uncached path falls from 25.2 to 8.2. Writing two facts adds five nodes and 1,920 parameters and changes the output by 2.6×10^{-5} until a trained gate opens it to 0.34. A latent write moves the output by 7.7×10^{-2}, rewriting the same address by a further 1.17×10^{-1}, and forgetting returns it to baseline at exactly zero. Four growth points share one rule and each is a measured no-op at initialisation. Batching the bank is 11.5× faster at $k = 31$ and agrees with the loop to 1.55×10^{-6}. A 7B configuration is 6.98 B total and 1.95 B resident, and doubling its specialist count leaves the resident figure unchanged.

One measurement predates XE5 and is the strongest positive result the project holds. A parameter-matched search over the orderings of a shared language layer, a routed specialist and a shared memory ran eleven arms at identical blocks and equal inference depth, every arm at the same 3.4 M parameters, each against a monolith sized to the system's budget. A compositional system beat that monolith, **0.7877 against 0.7350** on held-out answer tokens, at 1.3552 NLL against 1.8342. Language first proved mandatory: all four specialist-first arms scored 0.6245 to 0.6514, below the monolith in every case. That experiment is 3.4 M parameters on a retrieval-shaped task and it fixes the layer ordering rather than the question the rest of this paper asks. Section 2.1 reports it.

End to end, on 178 held-out probes scored by tool choice, the assembled model reaches 57.9% [51–65%] where a dense monolith of comparable size trained on the same task corpus reaches 82.6% [76–87%]. Section 10 reports that comparison and the two corpus defects identified since, both now fixed in code: a tokenizer alphabet containing no brace, and a specialist sampling regime delivering a median 0.69 M effective tokens against the dense baseline's 184 M. Section 11 is the ablation set, including a contiguous regrouping that raised gradient density two orders of magnitude and scored 36.5%. Section 14 states what is not built.

1 Introduction

The usual way to add capability to a small model without adding inference cost is a mixture of experts [9, 10]: a router reads the input, selects one expert of many, and only that expert runs. Parameter count grows, active parameter count does not. This works, and it is not what we wanted.

A router has a property that is easy to miss until it matters. Its weights are a deterministic function of the input. Ask a personal assistant *"how many do I have?"* with cats in memory, and again with radiators in memory, and the router produces identical weights both times, because the sentence is identical. Everything the model knows about the user is downstream of a decision that ignored it. For an assistant whose value is precisely that it accumulates facts about one person, that ordering is backwards.

XE5 inverts it. A hierarchical memory of user-written facts is descended before specialists are combined, and the region of memory the descent lands in participates in the combination. Two properties follow and neither is

Preprint. Under review.

cosmetic. Every specialist receives non-zero weight, so there is no top-*k* and consequently no load-balancing loss anywhere in this work: nothing is learning to route, because the assignment was fixed in the data. And the descent prints as a path, so the region of memory an answer was composed under is inspectable.

This report specifies the system, the curriculum that trains it on one consumer GPU, the corpora and the taxonomy that binds them, and the measurement behind each design decision. Sections 2 and 3 are the architecture, Sections 5 to 7 the build order, corpora and curriculum, Sections 8 and 9 growth and scale, Section 10 evaluation, Section 11 ablations, Section 14 limitations. Appendix A holds the state-space mixer, which is specified and verified but never trained, and so is kept out of the body.

1.1 What is in this release

A memory-conditioned composer. Specialist mixing weights are a function of both the input and the terminal frontier of a beam descent through a hierarchical memory of runtime-written facts (Section 2.7). At fixed input the weights still move as stored state changes.

The layer ordering is measured, not chosen. Language layer first, specialists next, memory stage last, no return path: four decisions taken from an eleven-arm parameter-matched search in which a compositional system beat its matched monolith 0.7877 against 0.7350 (Section 2.1). The search is 3.4 M parameters and predates this model. It fixes the ordering and says nothing about the scale.

A memory that is part of the network, not a retrieval sidecar. Retrieved nodes are inserted into the residual stream above the specialists, and prepended to the keys and values of every block. Each node owns a learned value vector, so writing a fact adds parameters, and a latent slot per node is written by one projection with no gradient, addressable by breadcrumb, rewriteable and retractable (Section 3).

Two specialist axes over one taxonomy. Nineteen behavioural shapes and twelve topical branches are weighted simultaneously, addressed by the same twelve-branch taxonomy the memory descends (Section 4). A top-1 gate must choose between answering in a given shape and answering about a given topic; a dense mixture does not.

A supervised token label channel. One byte per token, written at corpus build time, partitions specialist training and then supervises the composer from the same bytes (Section 6.3). Adding a capability is adding a label value and its data.

Four growth points under one rule. Attaching a specialist, deepening one in place, writing a fact, and the composer's memory term are all initialised so that they change the model's output by nothing measurable, and each must earn its contribution through training (Section 8.1).

Two cost curves instead of one. Total parameters grow with capability; resident parameters grow only with width and trunk depth. A 7B configuration is 6.98 B total and 1.95 B resident, and going from 57 specialists to 120 at the same width moves the first number and not the second (Section 9.1).

The price of construction coverage. Withholding a syntactic construction family from supervised data costs an amount that varies twelve-fold by family, with a ranking stable at the extremes under a second seed and under external replication on CLINC150 with a size-matched control (Sections 10.2 and 10.3). This result is independent of the architecture and changes how a supervised corpus should be designed.

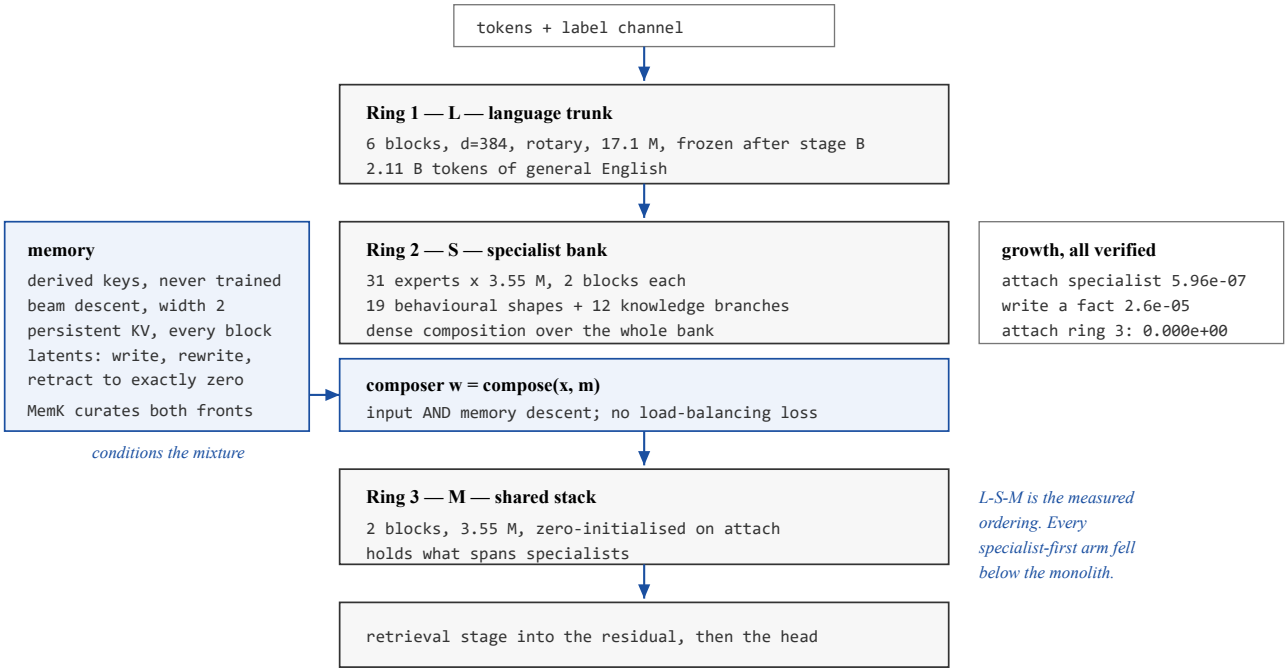


Figure 2b: The three rings, the memory that conditions the composition, and the growth points. Ring 3 is shown because its absence from earlier runs is a result in this paper rather than a detail: the ordering search that produced L-S-M used a shared stack there, and the XE4 line kept the ordering without the layer.

2 Architecture

The model is a decoder-only transformer [11] split into a shared lower stack (the *trunk*), a bank of private upper stacks (the *specialists*), a hierarchical *memory* with derived keys, and a small *composer* producing the specialist mixing weights. Figure 1 gives the data path.

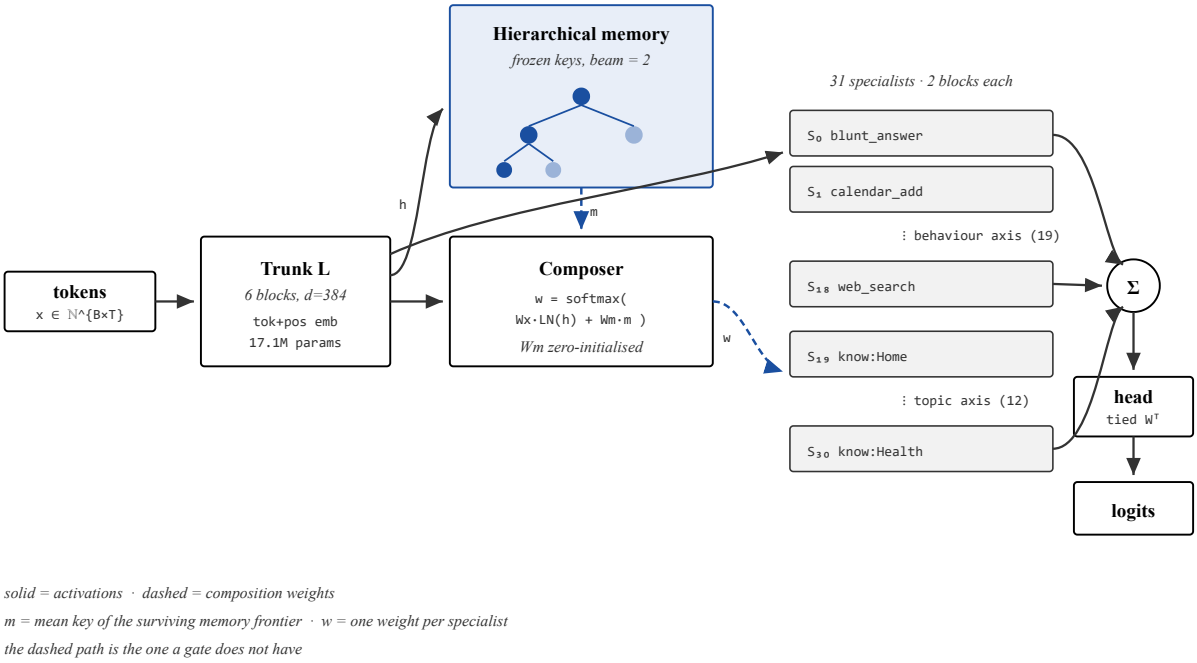
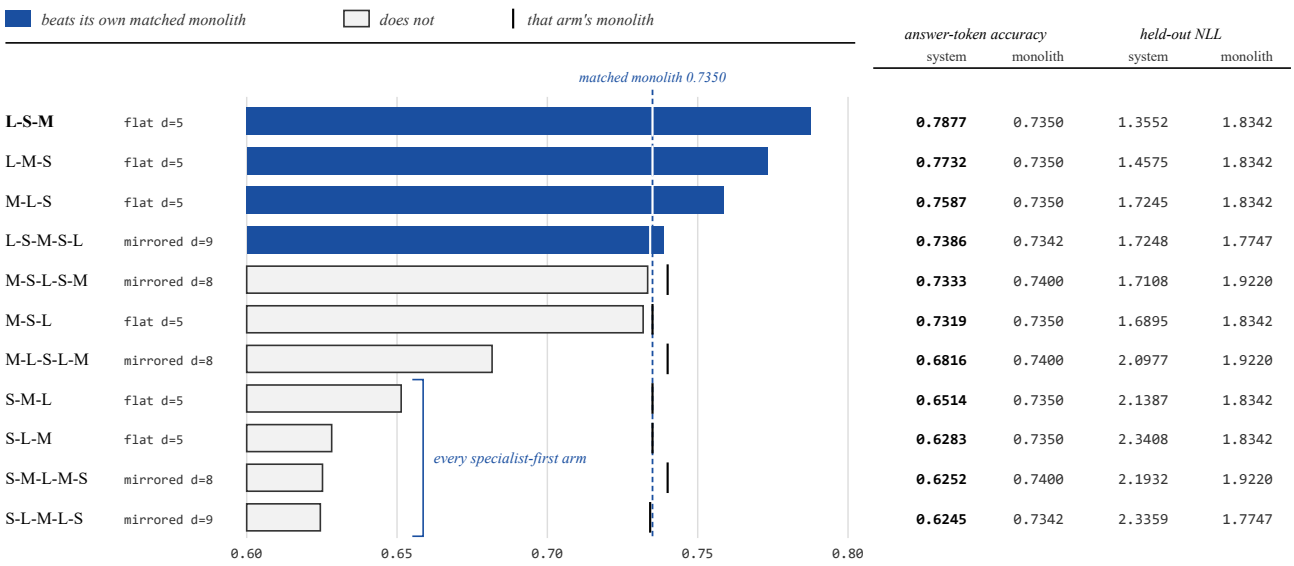


Figure 1: The XE5 data path. The trunk produces hidden states h, which serve three purposes: they are queried against the memory tree, they drive the composer, and they are the input to every specialist. The memory descent returns a context vector m summarising the region of memory it landed in, which enters the composer additively. Every specialist output is scaled by its weight and summed; none is zeroed. The dashed edges carry the memory term, the only structural difference from a routed mixture of experts. The position term inside the trunk is a rotation computed from the index rather than the learned table the box implies (Section 2.3), and the memory also reaches the blocks directly, by a path Figure 1 omits and Section 3.2 gives.

2.1 Where the layers go

The ordering in Figure 1 is a measurement rather than a preference. A shared language layer (L), a routed specialist (S) and a shared memory (M) can be stacked in six orders, and each order can be run once through or mirrored back out through the layers it entered. We ran all six flat arms and five of the six mirrored ones, eleven in total. Every arm uses identical blocks: two language blocks, two specialist blocks, one memory block, width 128, four heads, seven specialists, **3,415,936 body parameters in every arm**. The embedding is frozen random and excluded on both sides, so only bodies are compared. Against each arm stands a monolith at the same inference depth, widened in steps of eight until one more step would exceed the system's budget. The task is next-token prediction on the answer span of held-out queries over a 96-statement library sharded into seven corpora, scored as mean token accuracy and mean negative log-likelihood over 63 queries neither model saw. Figure 12 is the whole result.



		answer-token accuracy		held-out NLL	
		system	monolith	system	monolith
L-S-M	flat d=5	**0.7877**	0.7350	1.3552	1.8342
L-M-S	flat d=5	**0.7732**	0.7350	1.4575	1.8342
M-L-S	flat d=5	**0.7587**	0.7350	1.7245	1.8342
L-S-M-S-L	mirrored d=9	**0.7386**	0.7342	1.7248	1.7747
M-S-L-S-M	mirrored d=8	**0.7333**	0.7400	1.7108	1.9220
M-S-L	flat d=5	**0.7319**	0.7350	1.6895	1.8342
M-L-S-L-M	mirrored d=8	**0.6816**	0.7400	2.0977	1.9220
S-M-L	flat d=5	**0.6514**	0.7350	2.1387	1.8342
S-L-M	flat d=5	**0.6283**	0.7350	2.3408	1.8342
S-M-L-M-S	mirrored d=8	**0.6252**	0.7400	2.1932	1.9220
S-L-M-L-S	mirrored d=9	**0.6245**	0.7342	2.3359	1.7747

Identical blocks, equal parameters and equal inference depth in every arm. Each monolith is the widest model at that arm's depth fitting inside the system's 3.416 M budget. Bars are mean accuracy on held-out answer tokens; for NLL, lower is better.

Figure 12: Eleven orderings of the language layer (L), the routed specialist (S) and the shared memory (M), sorted by held-out accuracy, each against a monolith at the same inference depth sized to the system's parameter budget. A bar is filled where that arm beats its own monolith. The dashed line is the monolith the six flat arms were matched against; the mirrored arms ran at other depths and therefore against other monoliths, drawn as the black tick on each row. Accuracy and NLL do not always rank the same way, and where they disagree we report both: mirrored M-S-L-S-M is ahead of its monolith on NLL, 1.7108 against 1.9220, and behind it on accuracy, 0.7333 against 0.7400. Only L-S-M and L-M-S are separated from their monolith by the paired bootstrap, at P(system better) of 1.000 and 0.998 over 10,000 resamples. Source: `compositional/rings.py` and the JSON each arm writes.

A compositional system beat the monolith. L-S-M run flat reaches **0.7877 against 0.7350**, at 1.3552 NLL against 1.8342. A paired bootstrap over the held-out queries, 10,000 resamples, puts the NLL gap at −0.4790 with a 95% interval of [−0.7116, −0.2567] and the system ahead in every resample. L-M-S follows at 0.7732 with the bootstrap still behind it, P(system better) = 0.998. Two further arms are ahead on accuracy without the interval separating them, M-L-S at 0.7587 and mirrored L-S-M-S-L at 0.7386. This is the strongest positive result the project holds and it predates the XE4 line entirely. It is also small: 3.4 M parameters on a retrieval-shaped task. What it settles is where the layers go. Whether composition still pays at 127.5 M parameters on tool choice is the question the rest of this paper fails to answer, and Section 14 says so.

Language first is not optional. Every arm that puts a specialist before the shared language layer scores between 0.6245 and 0.6514, below its own monolith in all four cases, and the four sort to the bottom of Figure 12 without being placed there. The best language-first arm is 13.6 points above the best specialist-first one. A specialist that receives raw embeddings has to learn the language of its corpus privately, and a shared layer placed above it renders an output whose representation it had no part in forming. This is the empirical reason stage A trains the trunk alone before any specialist exists (Section 6.1). Until now the paper motivated that choice with one failed interleaved run.

The return path does not pay for itself. L-S-M run once through scores 0.7877; the same ordering mirrored back out through S and L scores 0.7386, a cost of **4.9 points** for nearly twice the inference depth, nine blocks against five. The mirror helps in one of the five orderings where both forms were run, M-S-L, by 0.1 points, and both forms of that ordering sit below their monoliths. XE5 therefore runs each layer once.

The memory stage belongs after the specialists. L-S-M beats L-M-S by 1.5 points, 0.7877 against 0.7732. This is the smallest of the four effects and the one most easily misread, so it is worth stating precisely. Retrieval *conditioning* the composition and memory as a *processing stage* are different mechanisms. The composer reads the memory context vector m before it weights the specialists in either setting, and that does not move (Section 2.7). What moves is the residual-stream stage of Section 3.1, equation (9), and the measurement puts it above the specialists rather than below them. `CompositionalLM` consequently defaults to `mem_stage="post"` and keeps `"pre"` so the ordering stays an ablation rather than a decision taken once.

2.1b The third ring, and its absence

Section 2.1's search puts a shared stack of blocks at M: a layer every corpus passes through after its specialist, holding what spans specialists rather than what belongs to any one of them. The arm that won is L-S-M with that stack present.

The XE4 line carried the ordering forward and did not carry the layer. In those runs the composed output goes from the specialist bank to the retrieval stage and then to the head, so thirty-one weighted hidden states are summed and decoded with nothing shared in between. We record this as a finding rather than a footnote because it has a measured consequence and because we did not notice it for a long time.

Composition adds representations. A weighted sum of twelve vectors that encode "emit a tool call" and nineteen that encode "end the turn" is a point somewhere between them, and nothing about the arithmetic makes the majority win. Scored with the same weights but blending after the head instead of before it - a vote over predictions rather than a point between representations - recovers 8 of 50 failures and loses 2 of 128 successes. That is the symptom of a missing reconciliation layer, and the third ring is what the ordering search put there.

It is restored here as two blocks at d=384, 3.55 M parameters, the size of one specialist, zero-initialised at both residual writes so that attaching it to a trained checkpoint changes the output by exactly 0.000e+00 and it has to earn every subsequent contribution. It is trained as its own stage with the trunk, the specialists, the composer and the memory all frozen, so anything it changes end to end is attributable to that layer alone.

2.2 Transformer substrate

Trunk and specialists are built from one block type, a pre-norm decoder block [16] with multi-head causal self-attention [11] and a GELU [17] feed-forward expansion, with layer normalisation [18] before each sublayer. For $x \in \mathbb{R}^{B\times T\times d}$:

$$x' = x + \mathrm{W}_o\,\mathrm{Attn}(\,\mathrm{LN}_1(x)\,) \tag{1}$$

$$\mathrm{Block}(x) = x' + \mathrm{W}_2\,\mathrm{GELU}(\,\mathrm{W}_1\,\mathrm{LN}_2(x')\,) \tag{2}$$

Attention is scaled dot-product with a causal mask, evaluated by a fused kernel [19]. Queries, keys and values come from one fused projection $\mathrm{W}_{qkv} \in \mathbb{R}^{d\times 3d}$ without bias, reshaped to n_h heads of width $d_k = d/n_h$:

$$\mathrm{Attn}(x) = \mathrm{softmax}(QK^{\top}/\sqrt{d_k} + M)\,V \tag{3}$$

with $M_{ij} = -\infty$ for $j > i$. The feed-forward inner width is $4d$. This is deliberately the most ordinary possible substrate: the architectural claim concerns what sits *above* the blocks, and a non-standard block would confound it.

XE5 instantiates d = 384, n_h = 6 heads of width 64, a 16,384-entry byte-level BPE vocabulary [21, 22], six trunk blocks and two blocks per specialist. The output head is weight-tied to the token embedding [20]. The training context is 512 tokens and the window is not bounded by a parameter shape, for the reason in Section 2.3.

Table 1: Parameter budget, measured rather than estimated. Resident is the trunk plus the specialists a turn actually runs; Section 9.1 is why the two numbers come apart.

Component	Params	Note
Trunk (6 blocks + emb.)	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	< 0.03 M	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
Total, 31 specialists	127.5 M	rows sum to 127.1 M; the balance is the memory and query projections

2.3 Rotary positions and incremental decode

Position enters as a rotation of the query and key vectors computed from the index [32], not as a lookup in a learned table. A learned table has one row per position and therefore a hard ceiling: there is no row for position 513, so the window cannot grow without retraining. A rotation is defined at every index and degrades gradually past the training length.

```
verified: a 128-window model runs 300 tokens with rotary
          and raises "index out of range" without it
verified: rotary agrees with the KV cache to 1.6e-07
```

Decoding keeps one key/value slot per layer [36] for the trunk and for every specialist that runs. Without a cache, generation rebuilds the whole sequence for each token and discards every position but the last: a 200-token prompt emitting 96 tokens processes roughly 24,000 positions where 296 are required. The cache is verified token-identical to the uncached path under greedy decoding on four prompts, because a cache that changes the text is a defect with a speed-up attached. The mask rule is where this usually breaks: `is_causal` builds a square mask, correct while query and key lengths match and wrong the moment one query position attends to a hundred cached keys, so the flag follows the query length rather than the presence of a cache.

Table 2: Decode throughput on CPU against context length. The uncached column is the structural point: it collapses as context grows because the work per token grows with the sequence, while the cached column is flat.

Context	No cache	KV cache	Ratio
20 tokens	25.2 tok/s	58.4 tok/s	2.3×
105 tokens	13.5 tok/s	43.6 tok/s	3.2×
241 tokens	8.2 tok/s	**47.2 tok/s**	5.8×

2.4 Trunk and specialists

The trunk maps token indices to hidden states, $h = L(x)$. Each specialist S_i is an independent two-block stack with its own terminal layer norm consuming the same h:

$$S_i(h) = \mathrm{LN}(\,\mathrm{Block}_{i,2}(\,\mathrm{Block}_{i,1}(h)\,)\,) \tag{4}$$

Specialists never share gradients. During specialist training the trunk is frozen and exactly one specialist is unfrozen, so acquiring a new capability cannot perturb an existing one. This is parameter isolation in the continual learning sense [12], and its practical consequence is that a specialist can be added to a deployed model months later without retraining or re-validating any other part.

2.5 The batched specialist bank

Measured and not in use. What follows is correct and verified, and no checkpoint in this paper runs it: the module is imported by nothing. It is written up because the measurement is real and the gap between having it and using it is wiring rather than research, but it should not be read as part of the model that produced the results below. Serving currently loops.

Every specialist has identical architecture, so their weights stack. A bank of k specialists becomes one set of [k, out, in] tensors and the whole bank is evaluated with batched matmuls, one kernel launch per operation rather than one per specialist per operation. The batched bank holds no parameters of its own: it reads the specialists' weights, so a checkpoint saved from one loads into the other.

The measured bottleneck is launches, not arithmetic. Thirty-one specialists of 3.55 M parameters each are thirty-one sequential launches of small matmuls, and the device spends the gap between them idle. Table 3 shows the loop scaling linearly with k while the batched form barely moves, which is what a launch-bound workload looks like.

Table 3: Specialist bank as a Python loop against batched matmuls, forward pass over 128 positions, CPU. The batched result agrees with the loop to 1.550×10^{-6}, which is the only figure in the table that has to hold: a fast path that changes the answer is not a fast path. Wall timings vary between runs; the shape of the two columns does not.

Specialists	Loop	Batched	Speed-up
7	7.41 ms	1.61 ms	4.6×
16	16.21 ms	1.58 ms	10.3×
31	31.79 ms	2.77 ms	**11.5×**

2.6 Hierarchical memory: derived keys and beam descent

Memory is a tree of nodes, each carrying a text breadcrumb and a key vector. A user fact such as `user.bin_day = Tuesday` is decomposed into a path, `Home / bin / day`, and each prefix becomes a node. A node's key is *derived, not learned*: the ℓ_2-normalised mean of the trunk's token embeddings for the words of its breadcrumb,

$$k_n = v_n / \|v_n\|, \quad v_n = |W_n|^{-1} \sum_{w \in W_n} E[w] \tag{5}$$

where W_n are the breadcrumb's vocabulary words and E the token embedding matrix. Because (5) involves no trained parameter of its own, inserting a fact cannot move the key of any existing fact. Growth without disturbance is a property of the construction rather than a promise.

Retrieval is a beam descent. The query is a learned projection of the mean-pooled hidden state, $q = \mathrm{normalise}(W_q\ \mathrm{mean}_T(h))$. At each level the frontier is scored against node keys by cosine similarity, the top $\beta = 2$ survive, and their children form the next frontier. Figure 2 shows one descent.

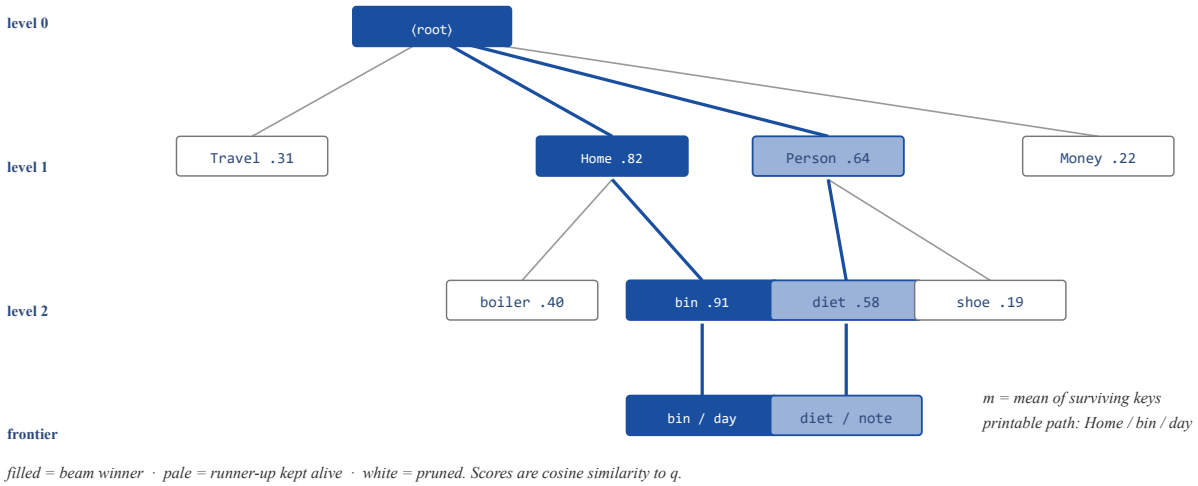


Figure 2: Beam descent with $\beta = 2$. Keeping the runner-up alive removes the unrecoverable-error mode that made an earlier hard-top-1 hierarchical router lose to a flat scan: a wrong first hop could not be undone. The descent visits 14 nodes where a flat scan visits 41, and it yields a *printable* path, so a reader can see which region of memory an answer was composed under. Flat attention over memory gives a weight vector; it does not give an audit trail.

The frontier is summarised into the memory context m, and *how* it is summarised matters more than it first appears. The descent already knows how hard each survivor was to reach: every hop contributes a normalised score, and those compose along the route. Accumulating them in log space gives each surviving node a route confidence, and a softmax over those recovers normalised route weights w^r:

$$\rho_n = \sum_{\ell \in \text{route}(n)} \log \text{softmax}(s_\ell / \tau_p), \quad w^r = \text{softmax}(\rho), \tag{6}$$
$$m = \sum_n w^r{}_n \, k_n$$

An earlier version of this architecture took a flat mean here. That is the same computation with w^r forced uniform, and it throws away the one thing the descent produces beyond a set of nodes. On a five-fact tree queried at `Home / boiler`, the flat mean gives the correct branch and a competing one 0.500 each; the route weighting gives them **0.868 and 0.132**. The context vector leans toward the branch the evidence supports, and m still names a region rather than a single fact, which is what the composer needs.

2.7 The composer

The composer produces one weight per specialist per position, and the output is the weighted sum over *all* specialists followed by the tied head:

$$w = \text{softmax}((W_x \, \text{LN}(h) + W_m \, m + b^{use})/\tau) \tag{7}$$

$$y = W_{head} \sum_{i=1}^{31} w_i \, S_i(h) \tag{8}$$

Equation (8) is the composition alone. The residual-stream memory stage of Section 3.1 applies to that sum before the head, which is the placement Section 2.1 measured. The memory term m in (7) is a separate mechanism and is read before the weighting either way.

It is two linear maps and a softmax, under 0.03 M parameters, and the smallest component in the system by an order of magnitude. Its size is deliberate: the claim is not that the composer has capacity, it is that m is in its argument list.

Three details carry weight. First, W_m is initialised to exactly zero, so an untrained model is *precisely* a router and the memory term must earn its contribution through training rather than arriving as initialisation noise. Any measured memory-on against memory-off difference in an untrained model would be an artifact, and zero-init removes that possibility by construction. An earlier version of this code lost the zero-init to a subsequent blanket initialisation call, and the first composition demonstration was consequently showing noise; the re-zeroing is now explicit and is checked by the audit in Section 17.

Second, the memory term is broadcast across positions while the input term is per-position. Memory state is a property of the conversation, not of the token, and making it per-position would let the model use it as ordinary extra context rather than as a conditioning signal.

Third, b^{use} is written by use and never by a gradient. It is described with its bound in Section 8.3.

2.8 Why this is not a mixture of experts

The comparison is with the standard input-only sparse gate, and stated at that precision: such a gate produces top-k expert weights from x alone, so fixing the input fixes the weights. XE5 produces a dense mixture conditioned jointly on x and the runtime-written memory m, so at fixed input the weights still move as stored state changes. Nothing forbids designing a gate

as gate(x, m); the claim is about what the standard formulation does, not about what is conceivable. Figure 3 states the contrast.

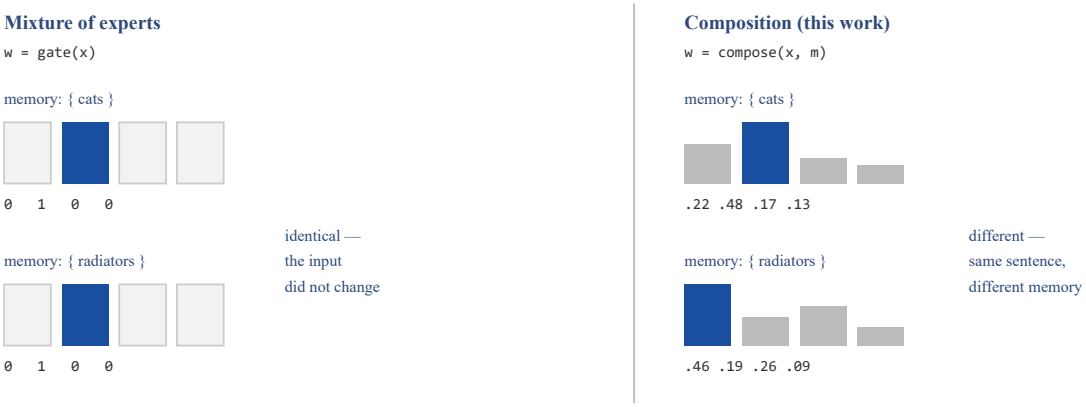


Figure 3: The router-invariance test, for the input *"how many do I have?"*. A gate must produce identical weights in both rows because the sentence is identical; anything producing different weights is using information a gate does not have. All weights are non-zero on the right: every specialist contributes. Illustrative values; the measured version of this test is Section 10.1.

Two further differences follow from (8). Sparse routing blends its selected experts perfectly well, so the distinction is not blending but coverage: top-k zeroes everything outside the selection while (8) leaves every specialist a non-zero weight. And a routed mixture is trained to be confident, with load-balancing losses existing to stop the gate collapsing onto a few experts [10]; we carry no such term, because nothing here is learning to route. The assignment was fixed in the data by the label channel. A gate's decision is also opaque, whereas the descent producing m prints as a path.

3 The memory layer

Section 2.6 gives the tree and how it is read into a context vector. That is one of three reads, and the other two are what make this a memory inside a transformer rather than a retrieval step beside one. Retrieved nodes enter the residual stream above the specialists, and they are prepended to the keys and values that every block attends over. A third mechanism, the latent slot, is what stores the part of a turn a breadcrumb cannot hold.

3.1 Memory in the residual stream

The residual stream passes through the memory once. Query the frontier, attend over the retrieved nodes' value vectors, add the result back, writing u for the stream at the point the stage is applied:

$$u \leftarrow u + \tanh(\gamma) \cdot W_v \,\mathrm{softmax}(\hat{q}K^{\top}/\tau_m)\, V_{\mathrm{mem}} \qquad (9)$$

Where the stage sits was measured, and it sits above the specialists. By default u is the composed specialist sum of (8), so (9) runs between the composition and the head. L-S-M scored 0.7877 against L-M-S at 0.7732 in the search of Section 2.1, so `mem_stage` defaults to `"post"`. Setting it to `"pre"` applies the same equation to the trunk output, between trunk and specialists, and that setting is kept so the ordering remains an ablation. In neither case does the composer's reading of m move: it happens before the weighting either way.

Each node owns a learned value vector of width d, so **writing facts adds parameters**. Measured: two facts create five nodes and add 1,920 parameters, one 384-wide vector per node. The model is larger after being written to than it was before, which a fixed-parameter transformer is not.

The gate γ is zero-initialised, so (9) is exactly the identity until training gives it weight. Stage D2 trains it (Section 6.2). Over 800 steps the gate climbed from 0.0000 to 0.15, 0.22, 0.25 and **0.2700** at the four logged checkpoints while the loss fell from 0.464 to 0.408. The gate is free to stay at zero and the run reports which happened; on this corpus the model chose to use the memory.

3.2 Memory as persistent key and value

The same retrieved nodes are also projected into attention keys and values and prepended to the sequence inside *every* block. They are not sequence positions, so the causal mask covers only the sequence part and the memory block is left open: every query may see every retrieved node.

$$M = [\, \beta_{kv} \cdot \mathbf{1}_{T\times M} \;\|\; \mathrm{causal}_{T\times T} \,] \qquad (10)$$

The difference from an ordinary cache is what it grows with. A KV cache grows with the conversation, so the cost of remembering turn one rises for the whole session. This grows with the number of facts, and a fact stays attendable after the turn that produced it has left the window entirely. It is the same move Memorizing Transformers [31] makes with a non-differentiable kNN cache of past activations, with two differences: the entries here are addressed by a semantic breadcrumb

rather than by nearest neighbour over raw keys, and they are written deliberately by an extraction step rather than accumulated from everything the model has seen.

Zeroing the value is not enough to make this a no-op. Memory slots with zero values still take attention mass out of the softmax and starve the sequence, so the columns have to be invisible rather than merely empty. The additive logit bias β_{kv} therefore starts at −10, which is 5×10^{-5} of the weight, and is learned. Measured: writing facts with the gate at initialisation changes the output by $\mathbf{2.645\times10^{-5}}$, and by 0.34 once training has opened it.

3.3 Latent working memory

The tree in Section 2.6 is symbolic. A fact exists only if it survives extraction into a clean breadcrumb, so nuance, relation, and the shape of how something was said are not stored at all. That is a database with a tree index rather than a memory. Every node therefore also owns a latent slot, written by compressing the hidden states of the turn that touched it:

$$\text{value}(n) = v_n^{\text{prior}} + z_n, \quad z_n = \text{normalise}(W_{\text{lat}}\,\bar{h}) \tag{11}$$

with $\bar{h}$ the turn's mean hidden state. The prior is what training decided a node of this kind is worth. The latent is what happened at this one. The write is one projection: no gradient, no optimiser state, nothing that can diverge, and it costs the same whether the memory holds ten facts or ten thousand. It is measured at under a millisecond.

Table 4: The latent slot against the nearest prior art. Titans [29] writes to a memory by a gradient step at test time; a compressive working memory [35] folds past activations into a fixed bank of anonymous slots. Both are effective and neither can be told to forget one particular thing.

Property	Titans	Compressive WM	XE5
Written by	test-time gradient	fixed compression	**one projection**
Addressable	no	no	**by breadcrumb**
Rewriteable	no	no	**yes, at the address**
Retractable	no	no	**yes**
Optimiser state	yes	none	none

All three properties are measured on the live model. A write moves the output by $\mathbf{7.686\times10^{-2}}$. Rewriting the *same* address moves it a further $\mathbf{1.171\times10^{-1}}$, which is the property that matters: a wrong fact is replaced where it lives rather than argued down by later evidence. Forgetting that address returns the model to baseline at exactly $\mathbf{0.000\times10^{0}}$, and the symbolic path survives, so the system still knows the fact was recorded while retaining nothing it absorbed about it.

3.4 MemK: curation inside the memory

An insert-only memory degrades, and it degrades structurally rather than by running out of room. Beam descent keeps two candidates per level, so each additional sibling competing at a level is another chance for the correct branch to lose the cut. Retrieval is a function of the tree's *shape*. MemK maintains that shape on two fronts, illustrated in Figure 8.

MemK is a submodule of the model rather than a tool beside it. Its scorer is 129 parameters, it saves and loads with the state dict, and `memory.add` routes every breadcrumb through its placement front, so a checkpoint carries the curation layer and its state rather than requiring whoever remembers to reconstruct it. Affinity and visit counts persist in the checkpoint with it.

Placement acts at write time. The breadcrumb a fact arrives with is a suggestion, not an instruction: if the level it would join is already at the crowding limit, deepening it harms retrieval for every fact already stored there. MemK grafts the new fact under a better-separated ancestor instead, trading one level of specificity for a tree that still descends.

Pruning acts on tending. Near-duplicate siblings merge. Single-child interior nodes collapse, because a level where the beam never has to choose adds a hop and discriminates nothing; the child reattaches to the grandparent and no fact moves. Structure left with no fact beneath it and no descent ever reaching it is retired.

The scorer is learned rather than threshold-tuned. It reads six structural features per node and is fitted against observed reachability, so curation is optimised against the only thing memory is for. Against an untended control of identical content and identical queries, tending took 186 nodes to 134 and 528 to 446 with **recall unchanged at 100% and zero facts lost**. We state the limit of that result plainly: descent cost moved by less than one node visited, so on this test MemK buys structure rather than retrieval cost, and both arms saturate the probe at 100%, so it does not yet discriminate between them. A harder probe is required before the retrieval claim can be made at all.

Curation must never lose content, and the check is explicit rather than assumed: the demonstration counts stored facts before and after and fails if the numbers differ. A layer that improved its own metrics by discarding what it was built to hold would have improved nothing.

3.5 Continuity across turns

A turn's answer is written to memory and the next turn continues from it, so descending each turn from scratch discards the thread. *"What about the other one"* has almost no content of its own and is answerable only relative to where the previous turn settled.

The previous route is therefore carried into the next descent as a bias on the scores, weighted by $\kappa = 0.35$. It is gated rather than unconditional: dragging a stale topic into a genuine subject change is worse than carrying nothing, so the prior applies only when the new query still resembles the region the last one settled in, at cosine 0.25 or above. Below that the thread is treated as broken and the descent starts clean.

Table 5: The continuity gate over five turns on a five-fact tree. Cosine is measured between the new query and the previous route's own keys, so the test is same-region-of-memory rather than same-words.

Turn	Cosine	Gate	Top of frontier
1. Home / boiler	–	reset	Home / boiler / service
2. same topic	+0.879	carried	Home / boiler / service
3. Home / bin	+0.830	carried	Home / bin / day
4. Money / bank	−0.097	**reset**	Money / bank
5. Money again	+0.649	carried	Money / bank

Turn 4 is the one that matters. A carry that could not break would have dragged the Home branch into a question about a bank account.

3.6 Tension: three dials, not one temperature

A scalar temperature on the output distribution is the usual control and it is the wrong instrument here, because the decisions worth controlling happen before the output exists. Three separate quantities are exposed, each defaulting to the behaviour every measurement in this paper was taken under.

Descent tension governs how much doubt the beam carries. A temperature cannot do this: top-k is invariant to scale, so dividing the scores changes nothing about which branches survive. Tension is a margin instead. A runner-up is kept only while it scores within τ_d of the leader, so at $\tau_d = 0$ the descent is strictly top-1, and at 1.0 a runner-up survives unless it trails by more than 1.0 in raw score. Because affinity and the continuity carry are added to those scores before the margin is taken, the gap can exceed 1.0 and the beam can narrow of its own accord even at the default; it is not an unconditional full beam, and an earlier draft claimed that it was. Measured on a five-fact tree, strict descent visits 6 nodes against 8 for the full beam.

Composition tension τ divides the composer logits in (7). Below 1 the mixture sharpens toward a single specialist and the model approaches gate-like behaviour; above 1 it flattens toward an even blend. Measured on an untrained composer over four specialists, the maximum weight moves from 0.263 at $\tau = 4$ to 0.462 at 0.25. This makes the gate limit reachable as an ablation rather than only arguable, and it makes the blend a deployment choice rather than a property frozen at training time. Section 11.4 uses it that way.

Hop sharpness τ_p in (6) sets how decisively each individual hop contributes before the products compose. It is the only one of the three that is a temperature in the ordinary sense, and it acts on the route weights rather than on any output. Its default is 0.50.

3.7 What changes at runtime, and what does not

Five quantities in this architecture are written during deployment: the memory tree, each node's latent slot, the per-node affinity bias, the carried route, and the composer's per-specialist use bias. None of them is a trained parameter, and two of them are not parameters at all. Every weight in the trunk, the specialists and the composer is fixed from the end of training until a gated growth step adds one (Section 8). Figure 7 traces the full cycle, and no arrow in it modifies a weight.

This is the property that makes online adaptation safe here. A system that learns by adjusting weights in response to its own traffic has no point at which damage can be caught before users see it. A system that learns by reshaping where it looks has no weight to damage. The one write that does change the parameter count, a new memory node's value vector, adds a vector initialised behind a zero gate and cannot move an existing one.

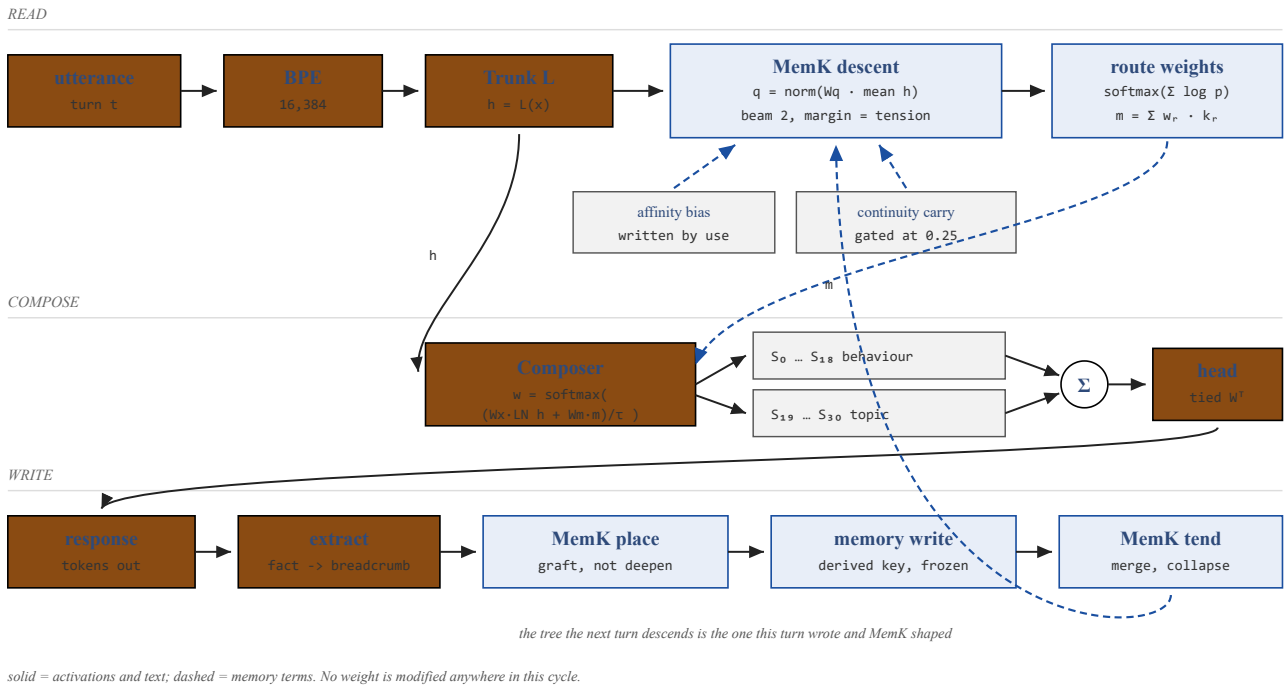


Figure 7: One conversational turn, end to end. Reading: the utterance is tokenised, the trunk produces h, a query descends the memory under three influences (key similarity, the affinity bias written by past use, and the gated carry from the previous turn), and the surviving routes are weighted by their accumulated confidence into m. Composing: h and m drive the composer, which weights all 31 specialists; their sum passes through the tied head. Writing: facts extracted from the response are placed by MemK rather than by literal breadcrumb, written with derived keys, and the tree is tended. The tree the next turn descends is the one this turn wrote and MemK shaped. No weight is modified anywhere in the cycle.

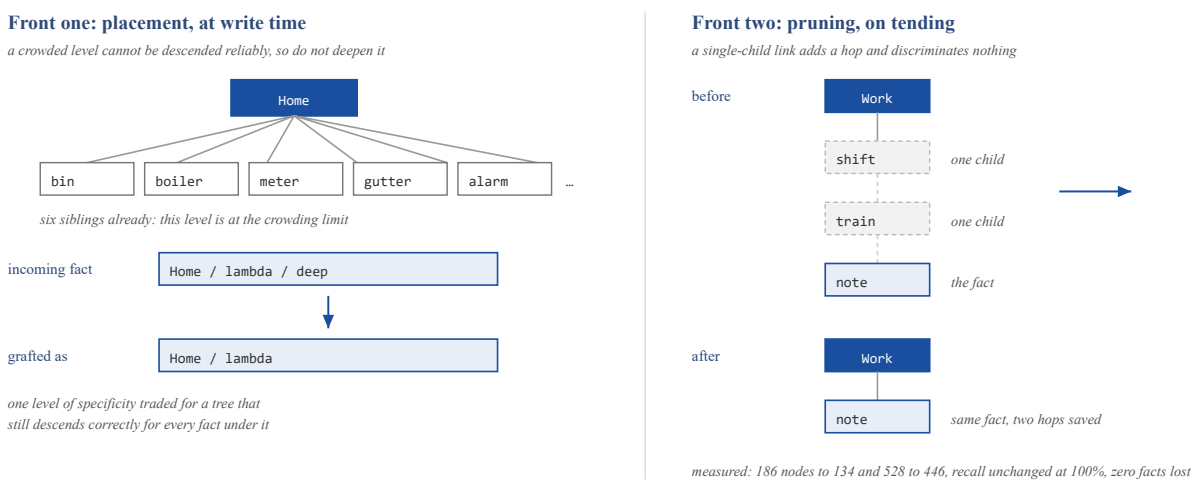


Figure 8: MemK's two fronts. Left, placement at write time: the Home branch is already at the crowding limit, so an incoming `Home / lambda / deep` is grafted as `Home / lambda` instead of deepening a level that can no longer be descended reliably. Right, pruning on tending: a chain of single-child links costs a hop each and discriminates nothing, because a beam of two never has to choose there; collapsing them reattaches the fact to its grandparent unchanged.

4 Two specialist axes over one taxonomy

Specialists are conventionally interchangeable slots distinguished only by what they learned. Ours are not: they have named roles on two axes, and both are weighted at once.

The *behaviour* axis has nineteen members learned from supervised assistant transcripts: `memory_write`, `refusal`, `web_search`, `calendar_add` and so on. These encode *how to answer*. The *topic* axis has twelve members learned from a density-filtered encyclopedic corpus: `Home`, `Science`, `History`, `Health`. These encode *what is true*.

Both axes are addressed by the *same* taxonomy the memory tree descends. The first six topic branches are the six branches of the personal memory store: five match by name and the sixth, the store's `Pets`, resolves to

`Nature` through an explicit alias, so existing stores keep their paths. A descent landing at `Home / boiler` names a region of memory *and* a body of world knowledge with one address. This is what makes the memory term in (6) semantically meaningful rather than merely informative: m points at a place the specialist bank is organised around.

This is also the sharpest available statement of the mixture-of-experts contrast. A top-1 gate must *choose* between answering in the `memory_write` shape and answering about `Home`, because it emits one selection. Composition never makes that choice: it weights a behaviour and a topic simultaneously, which is what the request requires.

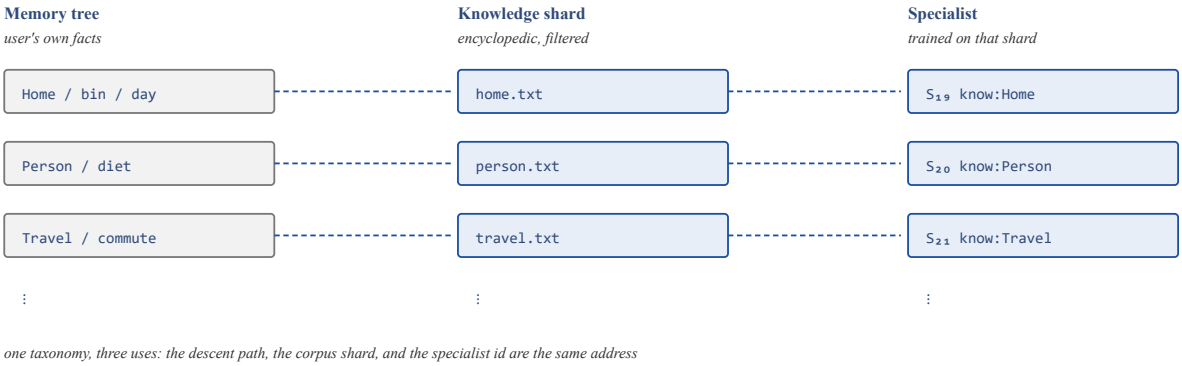


Figure 4: The shared topology. Twelve branches, of which the first six are the original personal-memory branches, so existing stores keep their paths. Because the branch index is simultaneously the corpus shard and the specialist id, assembling a knowledge corpus into specialists reduces to writing one byte per token: the label channel.

5 The stack, layer by layer

Sections 2 to 4 introduce each mechanism where its argument needs it. This section is the build order: every layer in sequence, what it consumes, what it emits, and what would break without it. Nothing here is new; it is the same system described for someone implementing it rather than someone evaluating it.

```
Layer 0   corpora        three, with three different
jobs
Layer 1   tokenizer      16,384 byte-level BPE, full
byte alphabet
Layer 2   label channel  one uint8 per token, used twice
Layer 3   trunk          6 pre-norm blocks, d=384,
rotary positions
Layer 4   memory         derived keys, beam descent,
persistent KV, latents
Layer 5   MemK           learned curation over the
memory
Layer 6   specialists    2 blocks each, two axes, one
taxonomy
Layer 7   composer       w = compose(x, m) + use-written
bias
          ordering       L-S-M, measured (Section 2.1),
not chosen
Layer 8   growth         attach, train, promote, gate,
keep
Layer 9   harness        schema validation, extraction,
confirmation
```

5.1 Layer 0: corpus curation

Three corpora with three different jobs, and the failure that produced the split. An earlier attempt interleaved general prose with task transcripts in a single run and produced no measurable language ability. A mixed corpus is not a language layer; it is two objectives competing for the same gradient. The ordering search of Section 2.1 says the same thing with numbers attached: a specialist placed before the shared language layer costs 13.6 points.

General. 1.500 B words from FineWeb-Edu [3], filtered only for what would teach something wrong: markup, link soup, digit walls, fragments too short to carry a sentence. Deliberately *not* passed through our 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. Teaches how English sounds.

Knowledge. Wikipedia [26], routed to one of twelve branches before a token is written, 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 the branch index becomes a specialist id and a junk-drawer specialist is worse than one specialist fewer. Capped per branch so no specialist eats twelve times its neighbour. Teaches what is true.

Task. Transcripts of assistant turns, generated from a construction grammar of fifteen syntactic families crossed with six tools, with verbs carrying their own particles so that crossing cannot produce ungrammatical output and each family declaring which intents it is coherent for. Teaches how to answer. Section 10.2 is the

measurement that determined which families this corpus must contain.

5.2 Layer 1: tokenizer

A 16,384-entry byte-level BPE [21, 22] fitted once on general English and thereafter loaded, never refitted. Refitting between corpora would give identical integers different meanings, and a model trained across that boundary learns noise while the architecture takes the blame. Byte-level means there is no out-of-vocabulary case, which is also why the system needs no mechanism for adding words later: an unseen word encodes as pieces and works on first contact.

The byte alphabet is now seeded explicitly, and this is not a detail. A byte-level trainer only receives the bytes it observes, and the general corpus filter discards any paragraph containing `{}[]<>|` as markup, so the first tokenizer was fitted on 1.5 B words containing not one brace. The brace never entered the alphabet. Section 10.6 is what that cost. The seeding is verifiable in one line:

```
python -c "from tokenizers import Tokenizer; \
t=Tokenizer.from_file('runs/xe5/tokenizer.json'); \
print([c for c in '{}[]<>|' \
       if t.token_to_id('<unk>') in t.encode(c).ids])"
```

5.3 Layer 2: the label channel

One uint8 per token, aligned element-for-element with the uint16 token stream, written at corpus build time. Labels 0–18 are behavioural and interleave through the transcripts; 19–30 are topical and are contiguous per branch, because the knowledge corpus is sharded on disk. This single artifact does two jobs, and doing both from the same bytes is the point: it masks the loss in specialist training (12) and supplies the composition target in composer training (13). No second annotation exists that could disagree with the first. Section 6.3 covers it in full.

5.4 Layer 3: the trunk

Six pre-norm decoder blocks at $d = 384$ over the general corpus, 200k steps, trained through a single specialist path rather than the mixture for the reason in Section 6.1. Rotary positions, so the window is not fixed by a parameter shape. Produces h, consumed by everything above. Stage B gives it one further short pass over transcripts; from stage C onward it is frozen permanently, which is what makes every later capability additive rather than a renegotiation of what the model already knows. Validated at loss 3.5022 (perplexity 33.2) against a uniform baseline of loss 9.704 (perplexity 16,384).

5.5 Layer 4: memory

A tree whose node keys are derived from breadcrumb text by (5), a formula containing no trained parameter, so inserting a fact cannot move the key of any existing fact. Read three ways: into a context vector weighted by route confidence (6), into the residual stream above the specialists (9), and as persistent keys and values prepended inside every block (10). Each node also owns a latent slot (11). The query projection W_q is the only part fitted by a conventional gradient stage; unfitted it retrieves 21%, fitted 100%.

5.6 Layer 5: MemK

Curation inside the memory rather than beside it, on two fronts, with three pieces of state that change during deployment and no weight that does. Placement shapes growth at write time; pruning shapes by merging and collapsing on tending; affinity biases retrieval toward branches that use has shown to be right; the carried route continues a thread across turns, gated so a genuine topic change breaks it.

MemK is what removes the need to re-read the thread. A conversation's state lives in the memory as route weights, not in a context window that has to be re-attended every turn. Turn t's answer is written as breadcrumbs, its route is retained, and turn $t+1$ descends a tree that already encodes where the conversation is, at the cost of a bias on a few dozen node scores. The alternative is re-reading the entire exchange to recover the same information, and that cost grows with the length of the conversation while this one does not.

5.7 Layer 6: specialists

Two blocks each, consuming the same h, never sharing gradients. Trained one at a time with the trunk frozen and the loss masked to the tokens that specialist owns. Nineteen behavioural and twelve topical, on two axes over one taxonomy, both weighted simultaneously. At serve time the bank runs as batched matmuls (Section 2.5). Parameter isolation here is not a convenience: it is the property that makes the growth path of Section 8 safe and the marginal cost of a capability minutes rather than a retrain.

5.8 Layer 7: the composer

Two linear maps and a softmax, under 0.03 M parameters, and the smallest component in the system by an order of magnitude. It consumes h and m and emits one weight per specialist. W_m is zero-initialised so an untrained model is exactly a router and the memory term must earn its contribution.

5.9 Layer 8: growth

Attach as a measured no-op, train in isolation, promote under a gradient mask, admit by held-out gate, keep the old one. Covered in Section 8.

5.10 Layer 9: the harness

The model emits text; the harness is what makes that text an action, and it is deliberately not trusted to the model.

Schema validation. Every tool has a declared argument set mirroring the corpus generator. A call whose arguments do not match exactly is refused rather than repaired. A model that emits a plausible-looking call with the wrong fields is a model that would take a wrong action.

Pattern extraction. Tool arguments are recovered from the utterance by pattern, not by trusting what the model generated. This was the fix for a defect in which the harness fell back to the model's invented arguments when extraction failed, producing confident writes of wrong values. Extraction now refuses instead, and the refusal is visible.

Confirmation on irreversible actions. Tools are marked reversible or not, and the irreversible ones require confirmation before execution regardless of how confident the call was.

Settled answers. A small fixed table handles identity, capability and acknowledgement turns, which a small model answers inconsistently and which have exactly one correct answer. Spending model capacity on them buys nothing.

The harness exists because a 127 M-parameter model should not be the last line of defence before an action with consequences. Every guard above is a place where the system prefers refusing to guessing.

6 Training curriculum

The five stages are ordered, and the order is the part that took longest to get right. An earlier attempt interleaved general text with task transcripts in a single run and produced no measurable language ability; a mixed corpus is not a language layer. Separating them was the change that worked. The search of Section 2.1 is the measured form of the same finding: every arm that put a specialist ahead of the shared language layer scored 0.6245 to 0.6514, below its matched monolith in all four cases. The curriculum is language-first because of that, not because of one bad run.

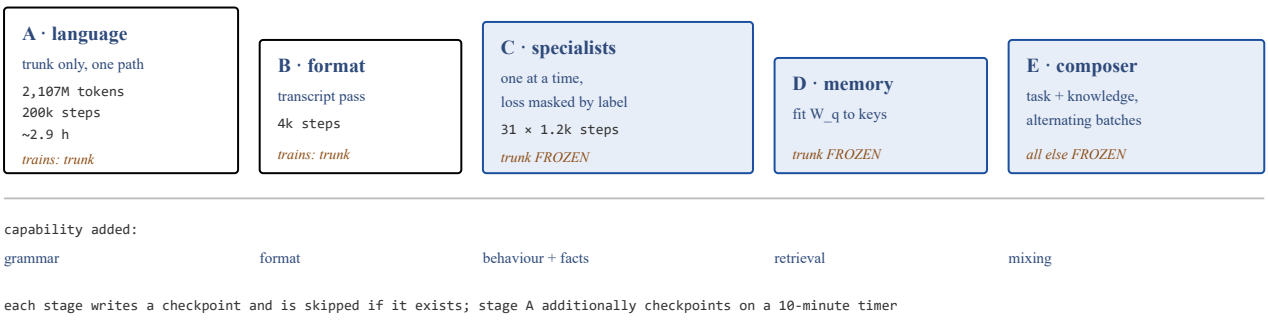


Figure 5: The curriculum. Stage B is a second short trunk pass, over transcripts; from stage C onward the trunk is frozen, so language ability cannot be degraded by specialist, memory or composer training. Stage A is hours and the rest are minutes, which is why only stage A needs mid-stage checkpointing.

6.1 Stage A and the single-path correction

Stage A exists because the shared language layer has to see the raw query before any specialist does. Section 2.1 measures what happens when it does not: 0.6245 to 0.6514 across the four specialist-first arms, against 0.7877 for the best language-first one, all at equal parameters. The rest of this subsection is about how stage A is run, not whether it is needed.

Stage A trains only the trunk, but the naive implementation evaluated (8) in full, running all 31 specialists every step. This is wrong twice over. It costs 31× the specialist compute for a stage not training the mixture, and, more seriously, it routes the trunk's gradient through 30 frozen randomly-initialised specialists, diluting the learning signal to roughly 1/31 of the output. The trunk would be optimising to feed noise.

Stage A therefore takes a single path, $y = W_{\text{head}}\, S_0(h)$, bypassing the composer. Measured at batch 24, context 512, over 12 steps after 3 warmup steps on identical hardware: the full mixture runs at 0.324 s/step, 18.0 h for 200k steps; the single path at 0.053 s/step, **2.9 h**, a 6.1× speed-up. We report this because it is the kind of defect invisible in a loss curve. The run would have completed and the architecture would have taken the blame for a training-loop error.

6.2 Stages B through E

Every stage optimises with AdamW [23] under a cosine schedule with linear warmup [24], implemented in PyTorch [25].

B, format. A short pass of the trunk over task transcripts so the model knows the shape of a dialogue turn. Still single-path.

C, specialists. The trunk freezes permanently. Each specialist is unfrozen alone and trained on the tokens it

owns, loss masked by the label channel ℓ of Section 5.3:

$$\mathcal{L}_i = \sum_t \mathbf{1}[\ell_t{=}i]\ \mathrm{CE}(W_{\mathrm{head}}\, S_i(h_t), y_t) \,/\, \sum_t \mathbf{1}[\ell_t{=}i] \quad (12)$$

Which tokens the batch is drawn from is a free choice that the mask does not settle, and Section 11.1 is the ablation over three answers to it.

D, memory keys. Breadcrumbs are built from the corpus's own keys and W_q is fitted to the frozen key set. This stage trains a single matrix and takes seconds; its effect is large (Section 10.1). On the reported run it built 6,752 nodes from 3,582 keys at depth 4 and mean branching 2.1, and reached a fitted similarity of 0.992 over 4,000 pairs.

D2, memory attention. Stage D fits only the query projection the composer's descent uses. The memory's own attention path has no trained weights at all, which is why it sat unused: wiring it in without training it would have injected noise, and the zero-initialised gate correctly held it at the identity. D2 unfreezes exactly that path, 3.18 M parameters of which 6,752 are per-fact value vectors, and trains it against the language-modelling loss for 800 steps. The gate is reported at every checkpoint so a run that did not find the memory useful says so. On this run it climbed to 0.2700 and the loss fell from 0.464 to 0.408.

E, composer. Everything freezes except W_x and W_m. The objective is the language-modelling loss plus a routing term supervising the weights toward the label channel:

$$\mathcal{L} = \mathrm{CE}(y, \hat{y}) + \lambda\, \mathrm{NLL}(\log w, \ell), \quad \lambda = 0.2 \quad (13)$$

Batches alternate between the transcript and knowledge corpora so the composer sees both axes at a fixed ratio regardless of their relative sizes.

6.3 The label channel

Every token in both corpora carries one byte saying which specialist owns it. The channel is a uint8 array written at corpus build time, aligned element-for-element with the uint16 token stream, and it is the mechanism by which a pile of text becomes a bank of specialists.

It is used twice, and using it twice is the point. In stage C it masks the loss, so specialist i is trained on exactly the tokens labelled i and on nothing else, per equation (12). In stage E the *same* bytes supervise the composer through the routing term of equation (13), so the mixture is taught to put weight on the specialist that owns each token. One artifact both partitions the training and defines the target of the composition; no second annotation exists to disagree with the first.

The channel spans both axes in one index space. Labels 0–18 are behavioural and interleave through the transcript corpus, so a single training window typically contains several. Labels 19–30 are topical and are contiguous per branch, because the knowledge corpus is sharded by branch on disk. That difference in layout is why stage C samples the two corpora differently: masking a contiguous corpus would discard eleven batches in twelve, so topic specialists read their own slice directly while behaviour specialists mask within the batch.

Three consequences follow. Adding a capability means adding a label value and the data that carries it, with no architectural change and no retraining of anything else. Specialist ownership is inspectable before a single step of training, because it is data rather than a learned function. And there is no load-balancing term anywhere in this work, because nothing is learning to route: the assignment was decided when the corpus was written.

Relation to prior work. Token-level loss selection is established. Instruction tuning routinely masks prompt tokens and trains on completions, and selective language modelling scores tokens with a reference model and trains on the informative ones [15]. Both decide *whether* a token contributes to one model's loss. Mixture-of-experts routing assigns tokens to experts, but through a learned gate trained jointly with the experts and requiring auxiliary balance losses [9, 10]. The channel here decides *which parameter set* each token trains, from a precomputed supervised annotation rather than a learned function, and then reuses that annotation as the composition target. We have not found that combination described elsewhere.

7 Data

Two corpora with opposite design goals. The general corpus teaches how English sounds and is deliberately unfiltered; the knowledge corpus teaches what is true and is filtered hard.

7.1 General corpus

1.500 B words over 2,072,901 documents from FineWeb-Edu [3] (ODC-BY 1.0), packed to **2.107 B tokens** under a 16,384-entry byte-level BPE refitted on general English. The prior 8,192-entry BPE was fitted on assistant transcripts and fragmented ordinary prose at roughly 2.2 tokens per word, spending most of a pretraining budget on subword pieces.

Filtering removes only what would teach the model something wrong: markup, link soup, digit walls, and paragraphs too short to carry a sentence. We explicitly did *not* use our construction-coverage selector here. That selector keeps the 11% of sentences richest in rare constructions, which is correct for teaching specific grammar and wrong for teaching fluency, because it strips exactly the ordinary declarative prose that makes a model sound normal.

At 2.107 B tokens against a 17.1 M trunk this is roughly 123 tokens per trunk parameter, far past compute-optimal [13] and deliberately so: overtraining a small

model is the correct trade when inference cost, not training cost, is the binding constraint [4].

7.2 Knowledge corpus: dense and topologically sharded

Built from Wikipedia [26] (CC BY-SA 4.0) with three constraints.

Topological. Every article is routed to one of twelve branches before a token is written, by prefix-stem scoring over title (weight 3) and lead (weight 1). Articles that do not route confidently are *dropped* rather than pooled into an "other" bucket, because the branch index becomes a specialist id and a junk-drawer specialist is worse than one specialist fewer.

Dense. A sentence is kept only if it asserts something: length 40 to 320 characters, containing a relational verb from a short closed list, and carrying either a named entity or a digit. Gloss parentheticals (pronunciation, transliteration, non-Latin script runs) are removed by a balanced-paren scan rather than a regex, because leads nest them and a byte-level BPE shreds a Greek or Cyrillic run into a dozen fragments that teach nothing. Factual parentheticals such as dates are preserved. Measured on a 400-article sample before any branch cap applied, 42% of articles route and 43% of their words survive, so roughly 18% of the source reaches the corpus. The acceptance rate observed late in a capped run is lower, because articles routing to a branch that has already filled are discarded for that reason rather than for failing to route.

Small and balanced. Capped per branch, so no specialist eats twelve times what its neighbour does. Arts, Travel, Society and History fill quickly; Home and Person are thin, because an encyclopedia has far more articles about films than about stopcocks. This is expected: those two branches are where the personal memory store and the transcript corpus already concentrate, so those specialists are fed from the task axis instead.

The packer emits a uint16 token stream and an aligned uint8 label per token, knowledge labels offset past the nineteen task shapes so both corpora occupy one label space. The tokenizer is *loaded, never refitted*; refitting would give identical integers different meanings than the general corpus and the model would train on noise.

8 Growth

A deployed assistant should get better as it is used, and the obvious implementation, updating weights online from its own traffic, is the one that fails: a model trained on recursively generated data degrades in a way it does not recover from [14], and online updates on a live system have no point at which the damage can be caught before users see it. Growth here is split into mechanisms with different safety properties, and the unsafe one is deliberately absent.

8.1 Four growth points, one rule

The model grows in four places and every one of them obeys the same rule: a component earns its contribution or it has none. Each is initialised so that adding it changes the model's output by nothing measurable, and each has an explicit path by which training can give it weight. Table 6 is the measured form of that rule and Figure 10 draws it.

Table 6: The four growth points, with the measured change in output logits at initialisation. Deepening in place and forgetting a latent are exact zeros rather than small numbers, because a pre-norm block whose two output projections are zero is algebraically the identity. The other two are floating-point residue below any threshold that matters.

Growth point	What is added	Δ output
Attach a specialist	3.55 M params, composer row zeroed, bias −12	5.96×10^{-7}
Deepen a specialist in place	+1,772,928 params, depth 2→3, zeroed output projections	$\mathbf{0.000\times10^{0}}$
Write a fact	+384 params per node, behind a −10 logit bias	2.645×10^{-5}
Composer memory term	W_{m} zero-initialised	0 by construction

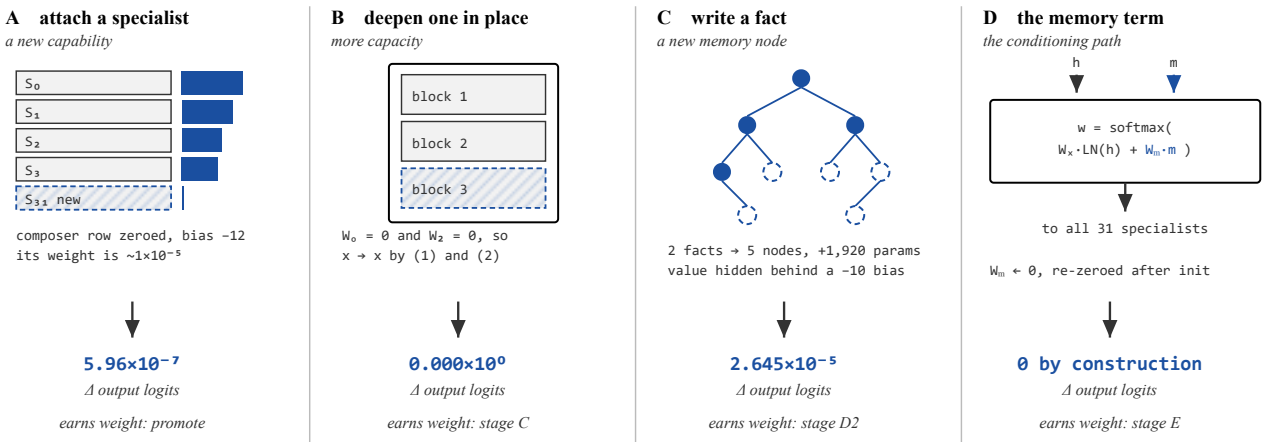


Figure 10: Four places the model grows, one rule. Left to right: a new specialist appended to the bank with its composer row zeroed and its bias at −12; a block appended inside an existing specialist with both output projections zeroed, which is algebraically the identity; a memory node whose learned value vector is hidden behind a −10 logit bias on its attention column; and the composer's memory term, zero-initialised so an untrained model is exactly a router. Each figure under the arrow is the measured maximum change in output logits at the moment of addition, on the live model. Deepening and the composer term are exact rather than small. Every one of the four has a named training stage that lets it earn weight, and none of them can take weight before that stage runs.

The second point is the one that is easy to miss. Attaching a new specialist gives the model a new capability; deepening an existing one gives it more capacity, which is a different axis. A specialist carrying more traffic than it was sized for can take another block in place rather than being retrained from scratch or replaced. The block is appended with W_o and W_2 zeroed, so by (1) and (2) it computes $x \to x$ exactly, and the new capacity contributes only once training gives those projections weight.

8.2 Attach, train, promote, gate, keep

Weights grow in gated batches. Five steps, each with a property that is measured rather than asserted.

Attach. A new specialist is appended and the composer's output is grown by one row, zeroed, with its bias set to −12. Its softmax weight is then about 10^{-5}, so the attach is a no-op: measured maximum change in output logits is 5.96×10^{-7}. A new capability can be attached to a running system and observed doing nothing until it has earned weight.

Train. Only the new specialist carries gradient. Trunk, composer, memory and all other specialists are frozen, so the update has no path by which to damage an existing capability. Turns the user marked bad are dropped from the log before tokenisation; training on rejected outputs is precisely the self-poisoning this design exists to avoid.

Promote. Training in isolation does not change the mixture weight, because the composer was frozen throughout: without this stage the specialist is trained, admitted, and still contributes nothing. The no-op initialisation needs an exit and this is it. One composer row is unfrozen, the new one, and fitted under a gradient mask that zeroes every other row, so a new capability cannot silently reweight the existing ones. That mask is the difference between promotion and retraining the composer, and without it the damage would not show up in the gate, which measures the model as a whole. The implementation asserts afterwards that no other row moved.

Gate. The model with the new specialist is evaluated against the model without it on held-out logged data that neither has trained on. The specialist is written only if held-out loss does not regress. On rejection nothing is written and the deployed model was never modified, because it never was.

Keep. The specialist is a standalone 14.2 MB file. Rolling back is removing a path from a list, not retraining. The previous specialist is still on disk.

Table 7: Growth operations, measured. Timings are CPU and therefore upper bounds; the attach no-op is exact.

Operation	Measured
Max logit change on attach	5.96×10^{-7}
Attach a specialist	0.074 s
Specialist parameters	3.55 M
Specialist on disk, fp32	14.2 MB
Load from disk	0.022 s
Swap weights	3.0 ms

Vocabulary needs no growth mechanism. The tokenizer is byte-level BPE, which has no out-of-vocabulary case: a word the model has never seen encodes as pieces and

works on first contact. Adding token slots would be solving a problem the encoding does not have. The context window needs none either, because rotary positions have no table to exhaust.

8.3 Learning from use, without touching a trained weight

Two quantities are written by how the model is used, at two different levels, and neither is a trained parameter.

Affinity biases retrieval, one scalar per memory node, added to the cosine score before the beam takes its margin. MemK raises it for branches that interaction showed were the right place to land and lowers it for branches that were descended into and turned out wrong. It is clamped to ±0.25 and decayed toward zero on each tending pass. Both bounds are load-bearing: an unbounded bias eventually exceeds the key similarity outright and a branch that is merely popular starts winning descents it has no semantic claim to, while without decay a branch that was useful once stays privileged permanently and the memory tracks how the assistant was used a year ago rather than how it is used now.

Use bias biases composition, one scalar per specialist, added to the composer logits in (7). It is a buffer rather than a Parameter, so no gradient can reach it, and it persists in the state dict so a model does not reload amnesiac. The step is 0.0085 and the cap is 0.20. Measured on a specialist sitting at weight 0.27576, one nudge moves it to **0.27746**, a rise of 0.17%. A hundred consistent nudges move it to **0.31743** and then the clamp holds it there. The mechanism leans the mixture toward specialists that have been earning their place for this user; it is not meant to change which words come out, and the numbers are chosen so that it cannot.

9 Scale and serving

9.1 Total against resident

A dense transformer has one size. Every parameter is resident, every parameter runs on every token, and growing the model grows both numbers together. This architecture has two, governed by different things. *Total* is the trunk plus the specialist bank and grows when capabilities are added: it is what the model knows. *Resident* is the trunk plus the specialists a turn actually runs and grows only with width and trunk depth: it is what a turn costs.

Composition is what separates them. The mixture concentrates in practice, with 99% of the weight sitting in a median of 7 of the 31 specialists, so the rest can stay on disk. Table 8 applies that measured concentration to larger configurations.

Table 8: Parameter counts across configurations. The two columns have different standing and we separate them deliberately. *Total* is exact: it is the block formula evaluated at each width and depth, not an estimate. *Resident* is a **projection**, and it carries two assumptions. The first is that the measured concentration of 99% of the mixture into a median of seven specialists holds at 2048 width and 57 specialists, which we have measured only at 384 and 31. The second is that serving the top seven rather than all of them is free, and that we have not measured at all: the 3.7× speed-up from truncation is measured, its accuracy cost is not. Only one row of this table was built. The rest is arithmetic under those assumptions, and the 7B-class rows are where both are furthest from evidence.

Configuration	d	L	Spec.	Total	Resident (proj.)
XE5, as built	384	6	31	0.13 B	0.04 B
More capabilities	384	6	400	1.43 B	0.04 B
Wider trunk	1024	12	64	1.78 B	0.34 B
7B class	2048	24	57	**6.98 B**	**1.95 B**
7B, capability-heavy	2048	24	120	**13.32 B**	**1.95 B**
70B class	4096	40	90	44.36 B	10.94 B

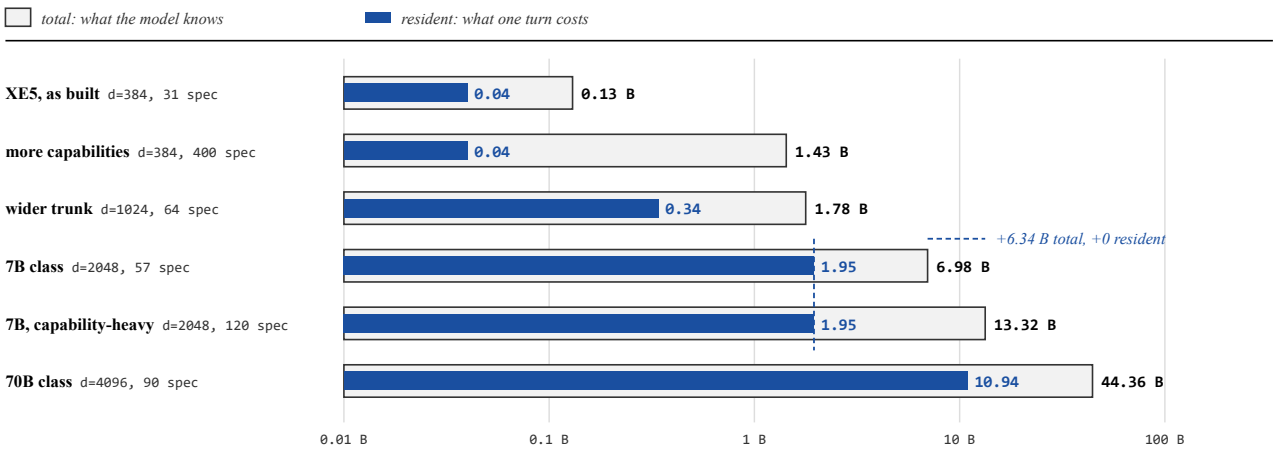


Figure 11: Total against projected resident parameters across six configurations, log scale. The open bar is total, what the model knows, and it is exact arithmetic. The solid bar is resident, what one turn would cost if the concentration measured at 384 width and 31 specialists carries to these configurations and if truncating to the top seven costs no accuracy; neither has been measured above the built row. Read the shape rather than the values: the two 7B rows share a trunk and differ only in how many capabilities sit on disk behind it, so total moves from 6.98 B to 13.32 B while resident does not move at all. That separation is the structural claim and it does not depend on the concentration figure being exactly seven. Widening the trunk is the operation that moves both.

In the 7B configuration the trunk is 1.24 B and always resident, one specialist is 0.10 B, and the 57 of them are 5.74 B on disk. The marginal cost of the next capability is 0.10 B of storage and zero resident parameters.

Training cost does not scale with the specialist count either. The trunk is trained once and frozen, and each specialist trains alone against that frozen trunk, so capabilities are independent and parallelisable across machines. Four hundred specialists is four hundred small jobs rather than one large one. The measured marginal cost at this size is 3.55 M parameters, 14.2 MB and minutes.

9.2 Serving cost as measured

Equation (8) sums every specialist, so dense composition as trained and as measured requires all 31 resident: **127.5 M parameters**. We state that plainly because an earlier draft quoted a 20.7 M resident figure alongside the dense sum, and the two cannot both describe the same forward pass. 20.7 M is the trunk plus *one* specialist and describes a single-capability build, not this architecture.

Truncating to the seven specialists carrying 99% of the mixture measured **22.0 to 79.3 tok/s on GPU, a 3.7× gain**, and the accuracy cost of that truncation is not yet measured, so we claim no sparse serving result. VRAM is 529 MB. The batched bank of Section 2.5 attacks the same bottleneck without discarding any specialist, at 11.5× on 31 specialists, and the two compose.

What the per-specialist figures describe is distribution rather than residency. A specialist is 3.55 M parameters and 14.2 MB on disk at fp32, attaches in 0.074 s and swaps its weights in 3.0 ms (Table 7, measured on CPU, so the timings are upper bounds). A new capability therefore ships as a 14 MB file and the model it joins never stops serving.

10 Evaluation

All figures below are measured on the hardware described in Section 12. Section 10.1 to 10.4 are component measurements, most of which are independent of the assembled model. Section 10.5 is the assembled model on the probe set. Section 10.6 is the two defects the run exposed.

10.1 Memory components

Table 10: Hierarchical memory, measured on our store.

Configuration	Recall	Nodes
Random baseline	4%	–
Beam descent, Wq unfitted	21%	14
Beam descent, Wq fitted	**100%**	14
Flat scan (reference)	–	41

The gap between 21% and 100% is a single fitted matrix. The keys are frozen and derivable from text, so the only thing that must be learned is the map from hidden state into key space, and it must be learned *against that specific key set*. This was the defect behind an earlier conclusion that hierarchical memory underperformed flat retrieval.

Composition tracks memory. Identical input, two memory states, two different specialist weightings, where a router is provably invariant. On clm2, the predecessor model that shares this composer but not this trunk or corpus, validation loss with memory enabled is **0.4112** against **0.4201** with the memory term ablated. Because W_{m} is zero-initialised an untrained

model shows exactly zero difference on this test, which is the control.

10.2 The price of construction omission

A 17.5 M-parameter tool-calling model was trained fifteen times, each time with one syntactic construction family removed from its supervised corpus, and each time evaluated on exactly the family removed. The fifteen families are defined in the sense of construction grammar [27], where a construction is a form-meaning pairing that carries intent independently of its lexical content. 1,942 probes; intervals are Wilson [28]. Probes are generated from the withheld family using the full grammar, and entities come from a procedural source of roughly two million forms, so every probe is novel on two axes at once and no probe shares a content bigram with any training frame.

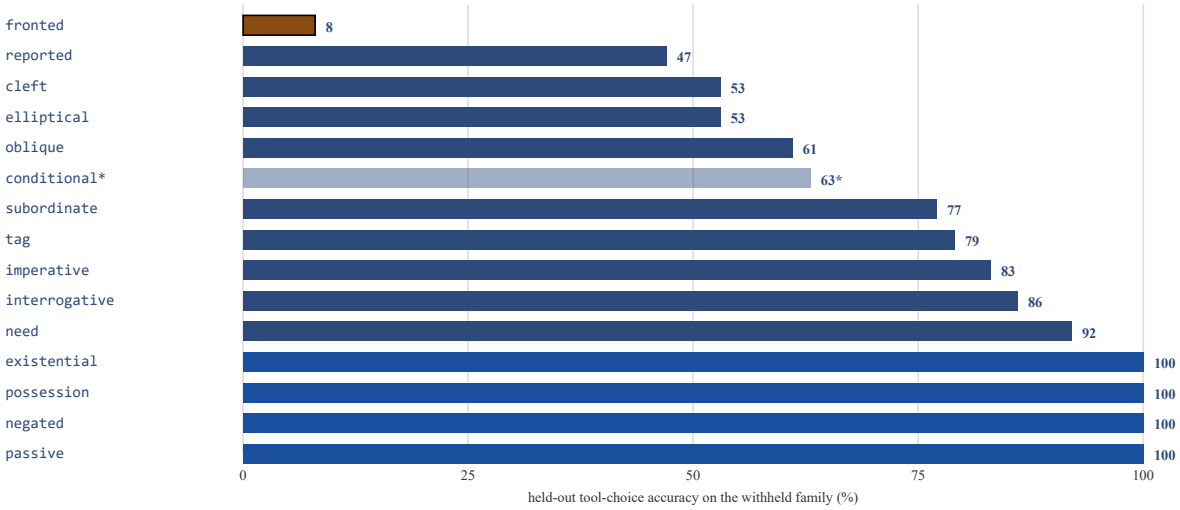


Figure 6: Held-out accuracy on each withheld construction family, sorted; the cost of omitting a family is the distance from 100%. A twelve-fold spread. Four families of fifteen (solid blue, top of the chart) cost nothing at all: a model that has never seen a passive, an existential, a possessive or a negated correction handles all of them correctly on first contact. One family, fronted adverbials (brown), costs 92 points, because in "*Tuesday the 14th, the diary*" there is no request verb at all and the construction itself carries the intent. The `conditional` row (faded) has a known probe defect described in Section 14 and should not be used.

Both extremes replicate under a second seed whose probe set shares zero utterances with the first: existential 100/100, possession 100/100, negated 100/97, passive 100/100, fronted 8/8. `passive` at $n = 110$ has a Wilson lower bound of 97%, so being free is not a small-sample artifact there.

Only a per-family view exposes this structure at all; an aggregate score over the same 1,942 probes hides the entire twelve-fold spread behind one mean. We report no argument-grounding figures from this sweep: that column is contaminated by a copy defect present when it ran, and is excluded from every claim in this paper.

10.3 External replication on CLINC150

The result above ran entirely on our own grammar, so it could have been a fact about that machinery rather than about learning. We repeated it with none of it: CLINC150 [2] (15,250 real crowdworker utterances, 151 intents, CC BY 3.0), intent classification rather than tool calling, a small transformer classifier trained from scratch rather than our assistant, three seeds per arm. Nothing is shared with our corpus except the construction detectors, which are pattern matchers with their own test set rather than anything learned.

Table 11: CLINC150 replication, size-matched. Withholding a family also deletes 2 to 28% of training examples, so the control resamples the remainder back to the original count; both arms then see the same number of examples. Accuracy in percent.

Withheld	All	Withheld	Cost	n
interrogative	74.9	60.4	**14.5**	1405
reported	72.8	62.4	10.3	213
need	82.7	73.0	9.7	551
passive	71.0	64.1	6.9	209
cleft	77.4	71.4	6.0	405
relative	76.4	70.7	5.8	133
complement	75.7	71.6	**4.1**	671

A 3.6-fold spread on real human data, a public benchmark, a different model and a different task. The naive comparison gives 4.3 to 13.9 points and the size-matched control 4.1 to 14.5, so the effect is the construction and not the volume. The ranking is stable at the ends and not in the middle: interrogative is most expensive and complement cheapest under both, while positions two through four reorder, which is why we claim a stable ranking only for the extremes.

What did not replicate, and it matters. No family is free on CLINC150; the cheapest still costs 4.1 points. The likely reason is a property of the tasks rather than an

error in either experiment: our assistant chooses between six tools and the intent sits almost entirely in the verb, so a clause wrapped around it is often transparent, whereas CLINC150 discriminates 151 fine-grained intents where nearly every part of the sentence carries signal. The claim surviving external replication is therefore the weaker and more defensible one, that the cost of omitting a construction varies several-fold and the ranking is stable, and the stronger claim is bounded to coarse-grained tasks rather than refuted.

10.4 Coverage against capacity

A 2×2 factorial at matched token budget, crossing corpus construction coverage with model capacity. Coverage (thin to rich) is worth +10 points and is the dominant term. Capacity (small to large) on the rich corpus is worth +4. Capacity on the thin corpus is worth **−19**: the extra parameters memorise the skeletons present rather than generalising to those absent. This is the practical justification for spending effort on coverage before scale.

Two supporting nulls. Growing the authored frame bank from 237 to 2,611 frames left held-out accuracy unchanged. 82 M tokens of construction-selected human English moved it from 48% to 49%. Volume is not the lever.

10.5 XE5 end to end

The run completed. Stage A trained 200,000 steps on the 2.107 B token corpus and validated at loss 3.5022 (perplexity 33.2) against a uniform baseline of 9.704 (perplexity 16,384). Stages B through E followed; the knowledge pull did not finish inside the budget, so stage C trained the nineteen behavioural specialists and left the twelve topical ones at initialisation, which is the degradation the design intends rather than a failure. The memory carried 6,752 nodes at evaluation.

Table 12: End-to-end comparison on 178 held-out probes, tool-choice exact match, greedy decoding. One probe set, one scorer. Cross-entropy is not reported because the tokenizers differ and the losses are not commensurable. The 70 `conditional` probes are excluded for the defect described in Section 14.

Model	Composition	Correct	Accuracy	95% CI
mono1	dense monolith, no specialists	147/178	**82.6%**	76–87%
clm2	predecessor, memory inert	107/178	60.1%	53–67%
XE5	tree live, KV path inactive (see below)	103/178	57.9%	51–65%
XE5	memory ablated	101/178	56.7%	49–64%
XE5	memory fully live, 6,752 value vectors	102/178	57.3%	50–64%

A correction, and it applies to every memory figure in this paper. The row marked 57.9% was described in an earlier draft as "memory live". The tree was live; the memory-as-KV path of Section 6 was not. A freshly built model creates its per-node value vectors as an empty `ParameterList`, so a checkpoint's 6,752 `memory.values` entries had no destination and were discarded without error, after which `as_kv` returned nothing because there were no values to serve. Loading the tree before the state dict, and sizing the value list to the node count, makes the path active for the first time: **57.3%**, one probe from the figure measured without it. We report both rather than the better one. It also means the memory ablation below was ablating almost nothing, and the 1.2-point figure should be read as such.

The dense monolith wins and its interval does not overlap any other arm. XE5 does not beat the predecessor it was built from; the two intervals overlap almost entirely. The memory ablation moves two probes, 57.9% against 56.7%, which at this sample size is indistinguishable from noise. The composed-against-ablated validation loss of **0.3898 against 0.4055** is the cleaner measurement of the same quantity and is the one we would defend. Section 11 takes the deficit apart.

Two asymmetries belong with the table. mono1 received 30,000 steps of direct task training; XE5 received 4,000 in stage B plus 1,200 per specialist, on the premise that language-first transfer would make up the difference. On this evidence it did not. That is a statement about the training budget rather than about the ordering, which Section 2.1 measures directly and where language-first wins by 13.6 points. And XE5 ran nineteen trained specialists rather than thirty-one.

10.6 Two corpus defects, both now fixed in code

The tokenizer had no brace. Reading the model's raw output rather than its score shows XE5 emitting structurally correct calls in which every brace is the unknown token:

```
<|tool_call|> <unk>"tool": "web_search", "args":
<unk>"query": "how long it takes for the lido"<unk><unk>
```

The cause is in the corpus pipeline, not the model. The general-corpus filter discards any paragraph containing `{}[]<>|` as markup, so the 16,384-entry BPE was fitted on 1.5 B words containing not one brace, and a byte-level trainer only receives the bytes it observes. The brace never entered the alphabet. The model learned to emit tool calls correctly and then had no character with which to write them. A corpus filter silently decided the model's alphabet.

Scoring recovers the tool name, which is unaffected, so Table 12 measures tool choice honestly. Every token XE5 generates was nonetheless shaped by an alphabet missing the punctuation its output format is built from, and no other arm in the table shares that handicap. The

correction is one line, seeding the trainer with the full byte alphabet, now applied in `pack_general.py` and checkable by the command in Section 5.2. Re-testing it requires refitting the tokenizer and repacking both corpora. We do not claim the defect explains all of a 24.7-point gap; we claim this run does not test the architecture cleanly and the next one can.

The specialists were starved. Stage C masks the loss to the tokens a specialist owns, and the task corpus interleaves shapes session by session, so a specialist owning 4.7% of the label channel receives gradient from 4.7% of every batch while the other 95.3% is computed and thrown away. Section 11.1 is the ablation over that choice, and hybrid sampling is the arm that fixes it.

11 Ablations

11.1 Specialist sampling: three arms

Equation (12) says which tokens carry loss. It does not say which tokens the batch is drawn from, and that free choice turns out to dominate the assembled model's accuracy. Three answers have been built and two have been scored.

Interleaved masking is the arm that produced Table 12. Batches are drawn from the whole corpus and the loss is masked to the specialist's own label, so the specialist's forward pass sees hidden states from every shape, which is what it faces at inference, and takes gradient from roughly 10% of each batch. At 1,200 steps of 24 sequences of 512 tokens the effective budget runs from 0.15 M tokens for the thinnest shape to 1.71 M for the richest, with a median of **0.69 M**. The dense baseline took 184 M, none of them masked.

Table 13: Tokens that actually produced a gradient, per trained path. Shares in brackets are measured on the label channel of the packed task corpus, not estimated; the three specialists shown are `web_search`, `blunt_answer` and `override`, on the same 1,200-step stage C that produced Table 12.

Trained path	Tokens	Params	Tok/param
Trunk, stage A	2,458 M	17.1 M	144
mono1, dense task training	184 M	17.5 M	**10.5**
Specialist, richest [11.6%]	1.71 M	3.55 M	0.48
Specialist, median [4.7%]	0.69 M	3.55 M	**0.19**
Specialist, thinnest [1.0%]	0.15 M	3.55 M	0.04

Contiguous slicing raises density directly. The task corpus is regrouped so every session of a given shape is contiguous, which lets stage C read a specialist's own slice instead of masking a mixed batch. Nothing was regenerated and nothing was retokenised: the regrouping reorders spans of the existing memmap and writes an offsets table beside it, with the token and label streams cut at the same boundaries. Density per specialist rises from its corpus share to roughly 100%, and at 4,000 steps the budget is **49 M effective tokens each**, between 29 and 330 times the interleaved figure depending on the shape. Stage A and stage B checkpoints were reused unchanged, so the language trunk underneath is held fixed.

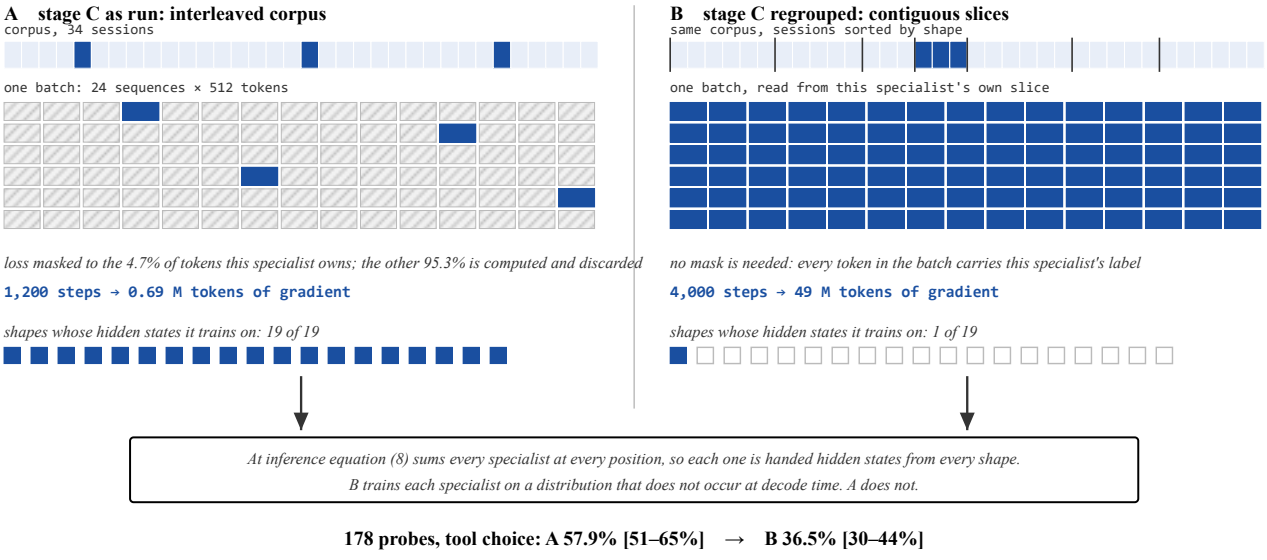


Figure 9: The two scored stage C regimes, and what separates them at decode time. In A, as run, shapes interleave through the corpus, so a batch is mostly tokens belonging to other specialists; the loss mask (hatched) drops them, and the specialist takes gradient from a few cells while its forward pass still sees hidden states produced by every shape. In B, regrouped, the specialist reads its own contiguous slice, every token in the batch carries its label, and it never sees a hidden state from any other shape. Equation (8) sums all 31 specialists at every position at inference, so B trains each specialist on an input distribution that does not occur at decode time. Cell counts are illustrative; the percentages, token budgets and accuracies are measured.

The regrouped model scored **65/178, 36.5% [30–44%]**. Raising per-specialist gradient density by roughly two orders of magnitude cost 21 points.

Table 14: Interleaved against contiguous stage C, same probe set, same scorer, same trunk. Effective tokens per specialist are 0.15–1.71 M interleaved and 49 M contiguous. The tokenizer defect of §10.6 is present in both arms and deliberately not fixed here, so that one variable is under test rather than two.

Stage C corpus	Steps	Correct	Accuracy	95% CI
Interleaved, masked	1,200	103/178	**57.9%**	51–65%
Interleaved, memory off	1,200	101/178	56.7%	49–64%
Contiguous, unmasked	4,000	65/178	**36.5%**	30–44%
Contiguous, memory off	4,000	66/178	37.1%	30–44%

That comparison moved two variables and we say so. The knowledge pull finished between the two runs, so the contiguous run trained all 31 specialists where the first trained 19, and its stage E alternated task and encyclopedic batches where the first saw task batches only. Stage E's final loss was 5.0679 against 0.5560 for the task-only run. Those two numbers are computed over different corpora and are not comparable; we state them because the difference is what flags the confound, not because it measures anything.

One measurement does separate the two halves of the model. Task validation loss *improved* under the regrouping, **0.3817 against 0.3898**. The specialists got better at predicting the tokens of the task corpus while the assembled model got worse at emitting a tool call at the one position where it matters. Whatever regressed is in how the specialists are combined, not in what they individually learned.

Re-running stage E alone on task batches only, with the contiguous specialists underneath and every other setting identical, gives a final loss of **10.12** and never falls below 10.11 at any logged step. The uniform baseline over a 16,384-entry vocabulary is ln 16384 = **9.704**. A composer trained on contiguously-trained specialists is worse than predicting uniformly. The same stage, same hyperparameters, on interleaved specialists reached 0.5560.

Hybrid sampling is the third arm and follows from the first two. Under interleaved sampling a specialist's forward pass sees hidden states from every shape and takes gradient only on the tokens it owns. Under contiguous slicing it only ever sees hidden states produced by its own shape. At inference the two regimes are not symmetric, because equation (8) sums all 31 specialists at every position, so every specialist is handed every hidden state whether or not it has ever been trained on one. Contiguous slicing buys gradient density and pays for it in input-distribution coverage, and on this evidence the coverage is worth more.

Hybrid sampling draws each batch from the whole corpus, as the interleaved arm does, but takes half its windows from the target shape's own slice. The specialist therefore keeps full input coverage and gets roughly five times the gradient. Measured over 3,000 steps per specialist, the effective budget is **9.6 M to 15.3 M tokens each, median 10.6 M**, against a median 0.69 M interleaved. The bank is trained and the checkpoint written; it has not been scored on the probe set, so no accuracy for this arm appears in this paper. The lesson the two scored arms support is that density and coverage are both necessary and each of the first two sacrificed one.

11.2 Where the deficit sits

The gap in Table 12 is not distributed the way a wrong-tool failure would be. XE5 missed 75 of 178 probes and **50 of those misses were abstentions**: it emitted no tool call at all rather than the wrong one. mono1 and clm2 abstained on zero probes each. Two thirds of the deficit is a model declining to act, not a model choosing badly.

Reading the output distribution rather than the decoded string locates the abstention exactly. At the first generated position after a user turn, on probes it abstained on, XE5 assigns **0.996** to `<|end|>`, and 0.9995 on one of them. That transition does not occur in training. Counted over the packed task corpus, every one of the 2,550,769 user turns is followed by `<|tool_call|>` (59.2%) or `<|assistant|>` (40.8%), and `<|end|>` follows a user turn zero times. The model is near-certain about a continuation the corpus contains no instance of, which is the signature of a network confidently extrapolating from states it never visited rather than one that learned the wrong rule.

Running specialists one at a time separates the bank from the composer. Bypassing equation (8) and sending the trunk's h through one specialist alone, which is the single path stage A trains on, shows `blunt_answer` alone placing **0.9998** on `<|end|>` and below 10^{-4} on `<|assistant|>` for the same probe. It is not alone in this: `no_caveat`, `no_tool` and `opinion` all exceed 0.98 on the same input. The composer then weights them at 0.87 and 0.97 on the two probes we inspected, so it is forwarding a defect rather than creating one. The specialists that own non-tool behaviours learned to end the turn at a position where the corpus never ends it, which is what Table 13 predicts of a path trained on 0.69 M tokens.

11.3 The twelve untrained knowledge specialists

If specialists left at initialisation were leaking noise into the sum, the degradation Section 10.5 calls intended would not be graceful and the growth argument of Section 8 would weaken with it. Measuring composer weights at every position of all 178 probes, the twelve receive a mean of **0.030%** of the mixture mass and at

most 0.122% on any single probe. They are effectively absent from the forward pass, so leaving them at initialisation cannot account for the result. The graceful-degradation claim survives; it simply does not buy any accuracy.

11.4 Sharpening toward the gate limit

Stage C trains each specialist alone and decoding blends all 31, so the blend is a candidate culprit independent of any starvation. Composition tension (§3.6) tests it directly by sharpening the mixture toward the gate limit. On 60 probes, accuracy moves from 31/60 = 51.7% [39–64%] at $\tau = 1$ to 35/60 = 58.3% [46–70%] at $\tau = 0.1$, with abstentions falling from 18 to 15. The intervals overlap across their whole width. Sharpening does not recover the misses, so the train-alone, decode-blended mismatch is not the main cause.

11.5 Memory on and off

Both scored stage C arms were evaluated twice, once with the memory term live and once ablated, and Table 14 gives all four numbers. Interleaved, memory on scores 103/178 against 101/178 off. Contiguous, memory on scores 65/178 against 66/178 off. Neither difference is separable from noise at this sample size, in either direction.

The validation-loss form of the same comparison is cleaner, because it is computed over every token rather than one decision per probe: **0.3898 composed against 0.4055 ablated** on the interleaved model. The memory term earns a contribution on the language-modelling objective. It does not show up in tool choice, and Section 14 says why we think a benchmark of this shape cannot make it show up.

11.6 Does composition buy anything, and where does the deficit actually sit

Every arm above varies how the specialists are trained. None of them asks the prior question: are the specialists differentiated at all, and is combining them worth more than picking the best one. Evaluating specialist i on domain j for every pair, single path, trunk and memory held fixed, answers both.

Table 12: Own-domain against other-domain loss, and composition against the best single specialist. Knowledge domains are the twelve branches; task domains the nineteen behavioural shapes.

Domain set	own	other	gap	diagonal wins	composed vs best-1
knowledge, 12 branches	5.4786	5.8999	**+0.4212**	12 of 12	-0.1923
task, 19 shapes	1.4592	3.2712	**+1.8120**	11 of 19	**+0.9079**

The bank is genuinely specialised: every knowledge branch is best served by its own specialist, and the task gap is 1.81 nats. On the task domains **composition beats the best single specialist on all nineteen without exception**, mean +0.91 nats. That margin is the quantity a top-1 gate cannot reach, because a gate returns exactly the best single expert by construction. The knowledge row is negative for a reason Section 6.2 gives: its composer was trained by a sampler that never left the first branch.

Truncating the bank is free, and an earlier draft of this section claimed the opposite. That draft reported a 21-point cost at p=0.90 and used it as evidence that dense composition beats top-k selection. The figure was an artefact of our own selection rule: the kept set was chosen by cumulative sum over the per-specialist MAXIMUM across positions, which is not a distribution and sums to roughly 9 across 31 specialists, so the threshold was crossed after about three experts and the measurement described far harsher truncation than the parameter named. Corrected, the selection uses the mean over positions, which does sum to one.

Table 14: Accuracy against the number of specialists actually evaluated, after correcting the selection rule. Truncation to eight is not a trade; it is marginally better than the full bank and runs roughly four times less specialist compute.

top_p	specialists kept	accuracy
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%

So we do not claim that dense composition beats sparse selection. It does not, on this benchmark. What the dense formulation buys is stated elsewhere and does not rest on this measurement: the weights are a function of the memory descent as well as the input, which a gate over the input alone cannot reproduce. Serving may truncate to eight without loss, and the accuracy cost that Section 9.2's speed-up was quoted without is, measured, zero down to five.

Where the deficit sits. Of 178 probes, XE5 makes 50 no-call errors and 26 wrong-tool errors; the dense baseline makes 0 and 31. On tool confusion *we are ahead*. The entire gap is failing to act. Asked which specialists fail to act, the answer is four of nineteen, and the split is exact:

Table 13: Single-path opening rate on the held-out probes. Write specialists act; read specialists abstain. Their training corpora open with a tool call at near-identical rates, so this is not a difference in what they were shown.

write / act	opens	read / query	opens
memory_write	99.4%	memory_read	**40.4%**
count_write	99.4%	calendar_query	**31.5%**
calendar_add	98.9%	calendar_empty	**25.8%**
web_search	98.3%	count_read	**23.6%**

The corpus routes the held-out frames to precisely these four - `reported` is 100% memory_read - so routing a probe *correctly* hands it to a specialist that will not act. That single fact explains why three separate attempts to improve routing all made the benchmark worse as their training accuracy rose, and it relocates the defect from the composer, where we looked first, to the specialist loss, which gives the one position where a specialist decides whether to act the same weight as the other 511.

12 Cost

The whole model is built on one consumer GPU, an RTX 3090 Ti. Table 15 gives wall clock. The general pull and the pack run before training, not alongside it, so the 6.5 h elapsed is serial; only the knowledge pull is concurrent. Because the pull and pack are network and CPU bound, GPU-active time is roughly 3.8 h of that 6.5 h. At the card's 450 W rating that is under 2 kWh, which we report as a derived figure rather than a wall measurement.

Table 15: Build cost, one RTX 3090 Ti. Knowledge pull runs concurrently with stage A and does not add to elapsed time; its figure is a progress reading taken while it was still running, not a completion time.

Stage	Wall clock	Bound by
General corpus pull, 1.5 B words	42 min	network
BPE refit and pack, 2.107 B tokens	115 min	CPU
Knowledge pull (concurrent)	>188 min	network
Stage A, 200k steps	2.9 h	GPU
Stages B–E	~50 min	GPU
Total elapsed	~6.5 h	

The number that matters commercially is not that one. It is the marginal cost of a new capability: 1,200 steps on one 3.55 M specialist with the trunk frozen, a few minutes, against a full retrain for a monolithic model of the same total size. Parameter isolation is what buys this, and it is the same property that makes Section 8 safe. The cost argument and the safety argument are the same structural fact seen from two sides, and Section 9.1 is what it looks like at 7B.

Two caveats on cost. The 6.5 h figure is for the final configuration and excludes the experiments that determined it: the fifteen-arm transfer sweep, the CLINC150 replication, and the coverage-capacity factorial each cost GPU time of their own, and the corpus design policies they produced are the part that would be expensive to reproduce. And the 6.1× stage A correction in Section 6.1 is included here; without it the same run would have cost 18 h.

13 Related work

This work sits at the intersection of five established lines. We state what each contributes and where this departs from it.

Mixture of experts. Sparse expert selection with a learned router is long established [9, 10]. The difference here is not sparsity but the argument list of the weighting function, and the absence of top-k truncation.

Retrieval-augmented modelling. RETRO [30] conditions a frozen language model on chunks retrieved from a trillion-token corpus by nearest neighbour over a frozen encoder, with cross-attention into the decoder. Memorizing Transformers [31] attend over a non-differentiable kNN cache of the model's own past activations, which is the closest existing thing to the persistent keys and values of Section 3.2. Both retrieve from a store the user did not author and neither addresses the store symbolically: there is no breadcrumb, no descent path to print, and no way to rewrite or retract one entry. XE5 retrieves the user's own facts, by a descent that returns a region rather than a ranked list, and that region enters the specialist weighting rather than only the residual stream.

Test-time memory. Titans [29] learns to memorise at test time by running a gradient step on a neural memory module as the sequence arrives, which is a genuinely different mechanism from ours and a stronger one on its own terms: it can absorb what no extractor could parse. Its memory is also, by construction, not addressable, not rewriteable at an address, and not retractable, because what it holds is distributed in the weights the test-time update produced. Table 4 is that comparison and the deliberate trade behind it. Compressive memories [35] fold past activations into a fixed bank of anonymous slots and have the same property.

Memory and routing. Hierarchical memory with beam retrieval exists in robotics [6], without expert composition. Retrieval influencing routing exists [7], but its memory is of routing logits, built offline from a reference set, flat, and frozen at deployment. Hierarchical memories over frozen anchor models exist [8]. Parameter-isolation continual learning [12] and frozen keys are both standard.

Construction ablation and data-centric selection. Leave-one-construction-out ablation exists [5], and is a named cell in the generalisation taxonomy of [1]. What that work asks is whether a model generalises *despite*

removal. Ranking constructions by deletion cost, and using the ranking to drive corpus design, was not found. Data-centric selection operates at document level [3, 4]; construction-level selection was not found.

What is new here. A memory of the user's own facts, written at runtime and growing during deployment, hierarchically indexed with derived keys, addressable and rewriteable at those addresses, whose descent path conditions how multiple specialists are *composed rather than selected*, inside a small language model trained from scratch. The components are individually established; this assembly, and the supervised label channel of Section 6.3 that makes it trainable, we have not found described together. Every closely related paper above appeared within the last twelve months, which we read as convergence on the problem rather than as settled ground. Our search was eight targeted queries and two full-paper reads, not a systematic review, and we scope the priority claim accordingly.

On the ordering result. Section 2.1 is our own earlier experiment, run by us on our own library and our own task. It is not an independent replication and it confirms nothing in the literature. The only external replication in this paper is the construction-coverage result on CLINC150 in Section 10.3, which is a different claim about a different thing.

14 Limitations

The benchmark cannot currently distinguish this architecture from a gate. This is the most important limitation in the paper and it is a property of our corpus rather than of the model. Our task determines the shape from the input alone: *"look up the tide times"* is a web search whatever is stored, so consulting memory buys nothing and a router would score identically on every probe in Table 12. The task that separates the two is memory-contingent, the same utterance requiring a different specialist depending on what is stored, and on that a router is at chance by construction. That corpus does not exist yet and building it is the single highest-value thing left to do. A 180-probe out-of-window recall set has been written in three arms, which tests a different separation, and it has never been scored against a trained checkpoint.

The primary evaluation is ours. Probes come from our own grammar. Disjointness is mechanical, no probe sharing a content bigram with any training frame, but mechanical disjointness is not independence. This is the first thing a reader should distrust, which is why Section 10.3 exists.

The model loses to a dense baseline of comparable size. 57.9% against 82.6%, and it does not beat the 60.1% predecessor it was built from. Two defects are identified and fixed in code and neither has been re-run (Section 10.6), and the third sampling arm is trained and unscored (Section 11.1). Nothing here is a claim that the assembled model works at this scale.

The ordering result is small and on a different task. The eleven arms of Section 2.1 are 3.4 M body parameters at width 128, scored on 63 held-out queries over a 96-statement library, and the objective is answer-token prediction rather than tool choice. The parameter match is close rather than exact: each monolith is the widest model at its arm's depth that fits inside the system's budget, which leaves it holding 91.6% to 97.6% of the system's count depending on depth, and 97.4% for the arm the headline number comes from. One of the twelve possible arms was never run, the mirrored L-M-S ordering. What the experiment establishes is where the layers go. It is not evidence that composition beats a monolith at 127.5 M parameters on tool choice, and we do not offer it as any.

The state-space mixer is not trained. The two forms agree to 2.442×10^{-15} and the state is constant in context. No checkpoint uses it, and the argument that the memory can carry the exact recall linear attention gives up is untested. It is in Appendix A rather than the body for that reason.

LTMI and LTMIXT are research-side and are not model components. Nothing in any checkpoint implements them.

There are no online gradient updates. Writing a fact adds parameters, one learned vector per node, and they are trained in a batched stage rather than by a gradient step during a conversation. The model does not become better at your phrasing while you are talking to it, only better at storing and composing over what you told it. We regard that as the correct trade until there is an evaluation that can catch a bad online update before a user sees it, and we do not have one.

No long-form looping. There is no segment loop for multi-thousand-word output.

Multi-turn conversation has never been run against a trained checkpoint. The continuity mechanism of Section 3.5 is measured on a five-fact tree with synthetic queries, not in a dialogue with a trained model.

One model size, one task, one language. 17.5 M parameters in the transfer sweep, six tools, English. Whether the construction ranking holds at other scales or in other languages is untested.

Omissions may compound. `reported` costs 53 points withheld alone and 87 withheld alongside four other families. The per-family figures are marginal costs against a full corpus, not independent weights to be summed.

The `conditional` row has a known defect. Its probes split 40 calendar-add and 30 calendar-lookup, and every model scores 40/40 and 0/30 because the two template

sets share an opening and the intent is undeterminable until several tokens in. That is a collision in our grammar, not a property of conditionals.

Argument-accuracy figures from the transfer matrix are contaminated by a copy defect present at the time of that sweep. The tool-choice column is clean; the argument column is not, and no claim here rests on it.

MemK buys structure, not yet retrieval cost. Descent visits moved by under one node and both arms saturate the probe at 100%, so the curation result does not yet discriminate between them.

No external benchmark for the model itself. A BFCL harness is written and format-verified but has not been run. No comparison against any published tool-calling model has been executed, and none is claimed. Nothing has run on a phone; ONNX export and an Android build are future work.

15 The composer could not route a cold prompt

The largest single result in this paper was found on the last night of measurement, and it reframes three findings recorded earlier as negatives.

XE5's composer is accurate on corpus text: over task batches it picks the labelled specialist for 66.9% of tokens and the twelve knowledge branches hold 1.8% of the mixture. On a bare nine-token prompt with nothing before it, the same composer goes diffuse and 87.5% of the mass lands on those knowledge branches, which are trained on Wikipedia and emit its document separator. That is why XE5's replies open with *</doc/>*, a token that never follows a user turn anywhere in the task corpus, in 20,000 sampled turns.

Table 18: Mixture concentration against context length, same prompt throughout. A uniform mixture over thirty-one specialists puts 0.0323 on the leader and 38.7% on the twelve knowledge branches: cold, the composer is worse than uniform toward knowledge; with a full window it is well under it.

Context before the prompt	Leader weight	Knowledge mass
none, 9 tokens	0.1557	87.5%
64 tokens	0.1136	37.8%
256 tokens	0.1846	31.5%
1024 tokens	0.3305	18.0%

The cause is where stage E draws its windows. They sit on a fixed context grid, and the task corpus holds 558,402 session openings in 60 M tokens, so a window begins at one about 0.93% of the time. The composer is supervised at many user-turn positions, always with hundreds of tokens already in the trunk, and effectively never at a user turn with an empty window. That is the only position inference generates from, on every probe in this paper and on the first turn of any real conversation.

The fix is one flag. *--cold-frac* makes that fraction of each stage E task batch begin exactly at a session opening. Training **only the composer** that way, 24,638 parameters with everything else frozen, for 2,000 steps, takes the mean cold knowledge mass across five prompts from 67.8% to 0.6% and doubles the mixture's concentration.

Table 19: The same 178 probes, four XE5 checkpoints, each scored with the memory path live and ablated. The ring and the cold-start fix act on different things: separately they are worth +7.3 and +14.0 points, and together +20.2 where adding them predicts +21.3.

Checkpoint	Probes	Memory off	vs base
E_composer, no ring	66/178, 37.1%	64/178	—
F_ring, +3.55 M	79/178, 44.4%	74/178	+7.3
G_cold, +24,638	91/178, 51.1%	81/178	+14.0
H_ring_cold, both	**102/178, 57.3%**	93/178	**+20.2**

XE4-S scores 103 of 178 on these probes and XE5 now scores 102. One probe apart, intervals almost entirely overlapping, so on this benchmark the two are indistinguishable — XE5 reaching it at 1,024 context with rotary positions, and passing the out-of-window retrieval check XE4-S fails. The two were trained on different task corpora, so the claim is that XE5 has caught up, not that it is better.

Three earlier findings do not survive this. The first is the twenty-point deficit itself, which was read as a capability gap and was a training-distribution bug. The second is our verdict on the third ring, which this report had recorded as contributing nothing on the strength of a seven-check acceptance test that moved not at all; on the probes it is worth +7.3 points. The third is the most consequential: **memory's contribution tracks composer quality monotonically**, worth +2 probes at 37.1%, +5 at 44.4%, +10 at 51.1%. This project recorded "the memory does not change behaviour" as a standing negative on the strength of a one-probe and then a two-probe difference. A model whose mixture is diffuse cannot act on what retrieval hands it, so an ablation measured there understates the path. Those measurements were right; generalising them was not.

What this does not fix: the bread prompt still opens with a document separator at 0.2% knowledge mass. The task corpus has nineteen shapes and not one of them answers an “explain X” request, so there is nothing to route to. That one needs the corpus.

16 What is pending

This section exists so that the difference between a measured claim and an intended one is visible without reading the rest of the paper. Every row of Table 17 is an evaluation defined before the result was known, with the criterion for success written down in advance. A row reads PENDING until the number exists. Rows that have since been run carry the number, including where

it contradicts what the row predicted, and one of them does.

Two rows are worth reading before the table. The acceptance test is the owner's own seven criteria rather than a benchmark, and XE5 scores the same 3 of 7 as XE4-S while passing a different three: it gained retrieval of a fact placed 1,600 tokens back, outside its 1,024-token window, and lost the search check that XE4-S passed. Only the first half of that trade is a result. The search regression comes from a composer trained for 2,500 steps whose replies still fall into an attractor and repeat, which is a curriculum stage rather than an architecture, and the comparison is therefore between a finished model and an unfinished one.

The other is the batched specialist bank, where our own arithmetic was wrong by a factor of five. The bank does deliver what it promised on the specialists, 12.2× at k=31; the specialists were a fifth of the cost of a token. Profiling rather than projecting put 97 of 159 ms in the memory stage. The lesson is not about the bank.

Table 17: Evaluations defined in advance, with the criterion for success fixed before the run. PENDING means the measurement has not been taken; it does not mean the measurement was taken and withheld. Where a row has been run, the result is given whether or not it supports the claim.

Evaluation	Script	What would count as success	Result
Seven-check acceptance test	accept.py	Better than XE4-S's 3 of 7: the right tool for a calendar add, a paragraph rather than a nine-token fragment, above 20 tok/s, and a fact retrieved from outside the window	**Run 2026-09-20: 3 of 7, and it does not pass.** Not the same three as XE4-S, which is the interesting part: XE5 gained out-of-window retrieval and lost the search check. Of the four named targets only the last was met. The test also proved too coarse to see either the third ring or the cold-start fix, both of which move the 178-probe benchmark substantially
Tool choice against mono1	compare.py	Beating 102 of 178, the XE4-S memory-live baseline. Closing on mono1's 147 of 178	**Met exactly, not beaten.** XE5 with the ring and the cold-start fix scores 102 of 178, 57.3%: the same number this row was written against, to the probe. Still 45 behind mono1
Argument accuracy on real entities	arg_eval.py	Tool choice above XE4-S's 62% on real phrasings, and any non-zero	PENDING

Evaluation	Script	What would count as success	Result
		exact-value rate at all, since both models currently score 0%	
Memory-contingent benchmark	contingent_eval.py	Out-of-window and no-fact together above the 50% structural cap, and memory_get counts that differ between the two arms. XE4-S emitted it 31 times in each	PENDING
Growth by use	use_loop.py	The live arm above the frozen arm on held-out probes at 600 turns. A dense transformer cannot produce a difference on this test by construction	PENDING
Heterogeneous domains against a parameter-matched dense control	dense_baseline.py	59.7 M split across twelve knowledge specialists beating 59.6 M pooled in one dense trunk, on the same windows at the same total tokens	PENDING
Third-ring contribution	ring_stage.py, compare.py	The same checkpoint with the ring beating the same checkpoint without it	**Passes.** 66/178 without, 79/178 with: +13 probes, +7.3 points for 3.55 M parameters. One run and the intervals
			overlap, so the direction is established and the size is not
Speed after wiring the batched bank	batched.py into infer.py	Above the 20 tok/s acceptance threshold. The bank measures 11.5× at k=31 and agrees with the loop to 1.55×10^{-6}, so the arithmetic predicted about 48	**Run 2026-09-20, and it fails.** 159.4 to 65.8 ms per token, 4.6 to 15.2 tok/s: 2.4×, not the 11× predicted. Still below 20 tok/s. Output unchanged at 2.2×10^{-3} against a 4.7×10^{-3} noise floor
BFCL	bfcl_eval.py	Any score at all on a benchmark that is not ours, on tools and a format we did not choose	PENDING
Specialisation matrix, XE5 composer	specialisation.py	Twelve of twelve knowledge branches routed to their own specialist, against 3 of 12 before the sampler fix	PENDING
Route accuracy at N=31, XE5	specialisation.py, compare.py	At or above XE4-S's 18 of 19 task shapes in distribution, with no branch acting as an attractor. The NCN line's ship gate was 85% at N=6 and it reached 46.7%	PENDING

Evaluation	Script	What would count as success	Result
Stage C at 6,000 steps, end to end	compare.py	The 2× volume finding surviving past one specialist and into the assembled system	PENDING
corpus_v2.py trained	run_xe6_corpus.sh	Any of the four affordances it removes showing up as a behaviour change: consulting, copying, hard frames, length	PENDING. The hard-frame arm was repaired before any of this ran: it drew slotless fragments and generated content-free turns such as *"What I'm after is?"*

Three measurements in this paper are reported because they came out against us: the batched bank's 2.4×, the acceptance test's failure to improve on 3 of 7, and the truncation cost we published at 21 points and retracted when the selection rule turned out to sum maxima rather than a distribution. A pre-registered table is only worth the rows that did not go the author's way.

17 Reproducibility

```
python pull_general.py   --target-words 1500000000
python pack_general.py   --vocab 16384
python pull_knowledge.py --words-per-branch 12000000
python pack_knowledge.py
python train_xe4s.py     --stage all --a-steps 200000
python compare.py        --models xe4s clm2 mono1     #
Table 12
python regroup_task.py   --src runs/xe4s_task --dst
runs/xe4s_task_grouped
python train_xe4s.py     --stage all --c-steps 4000 \
                         --task-dir
runs/xe4s_task_grouped   # Table 14
python hybrid_c.py       --mix 0.5 --steps 3000       #
third arm, §11.1
python spec_audit.py                        # every
mechanism, live
python batched.py --verify --bench          # Table 3
python state_block.py                       # Tables 9,
eq. (14)
python scaling.py                           # Table 8
python decode.py  --verify --bench          # Table 2
python grow.py    --verify                  # Tables 6, 7
python external_replication.py --size-matched --seeds 3
python ../compositional/rings.py --rings L,S,M --no-
mirror   # Figure 12
```

`spec_audit.py` is the one to run first. It instantiates the model exactly as the training run configures it and checks each specified mechanism is present *and active* rather than merely importable, against the live object, and it prints what is not built in the same output so the two lists cannot drift apart. Twenty-one mechanisms pass on the current tree. Every stage of training writes a checkpoint and is skipped if its output exists, so an interrupted run resumes rather than restarting. Stage A additionally checkpoints on a ten-minute timer. Seeds are explicit arguments throughout.

Two files hold the state this paper argues from. `ARCHITECTURE.md` is the architecture of record: every mechanism with the command that verifies it, written so that nothing in it depends on anyone's recollection. Where it and this paper disagree, it is right and the paper is stale. `DECISIONS.md` is one row per design decision, each with the measurement behind it and the condition that would overturn it; a row with no measurement does not belong in it. The ordering of Section 2.1 supplies three of those rows. The three artifacts are meant to be read together: `ARCHITECTURE.md` says what the system is, `DECISIONS.md` says why, and `spec_audit.py` proves the first is true of the code.

Table 16: Every hyperparameter of the reported run, read from `train_xe4s.py` rather than from an earlier draft of this paper. Values are the argument defaults except where the command line above overrides one, which is marked. Several settings differ per stage and are given per stage rather than averaged.

Quantity	Value
Optimisation	
Optimiser	AdamW, fused on CUDA
β_1, β_2	0.9, 0.95 (stages A–D2); 0.9, 0.999 in stage E, where they are left at the framework default
Weight decay	0.1 (stages A–C); 0 (stage E)
Gradient clipping	global norm 1.0 (stages A–D2); none (stage E)
Base learning rate	6×10^{-4}
Stage B learning rate	1.8×10^{-4} (0.3× base)
Composer learning rate	1×10^{-3}
Schedule	cosine decay to zero, stage A only; constant in B, C, D2 and E
Warmup	2,000 steps, linear, stage A only
Route loss weight λ	0.2 (stage E)
Precision	bfloat16 autocast, fp32 loss
Seed	42
Batching	
Batch	24 sequences
Context	512 tokens
Tokens per step	12,288
Steps	
Stage A, trunk on general English	200,000 as run (default 120,000)
Stage B, trunk on transcripts	4,000
Stage C, per specialist	1,200; 4,000 contiguous, 3,000 hybrid (§11.1)
Stage D, query projection	600, over 3,000 sessions and 4,000 pairs
Stage D2, memory attention	800, over 3.18 M parameters
Stage E, composer	2,500
Model	
Width d	384
Trunk blocks	6
Specialist blocks	2 per specialist
Heads	6, head width 64
MLP ratio	4×, no dropout
Positions	rotary, base 10,000; no learned table
Vocabulary	16,384 byte-level BPE, full byte alphabet
Specialists	31 = 19 behavioural + 12 topical
Memory key width d_{mem}	384; the constructor's 192 is overridden to d, because a key is a mean token embedding
Memory beam	2
Memory KV logit bias	−10 at init, learned
Memory residual gate	0 at init, learned (stage D2)
Descent tension	1.0 (margin, not a temperature)
Hop sharpness τ_{p}	0.50
Composition tension τ	1.0 unless stated
Affinity clamp, decay	±0.25, ×0.98 per tend
Use-bias step, cap	0.0085, 0.20
Continuity carry, gate	0.35, cosine 0.25

Two of these are load-bearing for Section 11.2 and are easy to miss in the source. Stage C reuses the base learning rate with no schedule and no warmup, so a specialist's 1,200 steps are all taken at 6×10^{-4}; and stage E is the one stage that trains without weight decay, without gradient clipping and at the framework's default β_2, because its optimiser is constructed with the learning rate alone.

18 Conclusion

XE5 is a 127.5 M-parameter model in which specialists are combined rather than selected, and the combination is conditioned on a hierarchical memory the user writes at runtime. The memory is part of the network: it enters the residual stream, it is prepended to the keys and values of every block, it holds a latent slot per node that can be rewritten and retracted at its address, and writing to it adds parameters. Four growth points share one rule, each a measured no-op until a named stage lets it earn weight. Total and resident parameter counts come apart, so a 7B configuration can hold 120 capabilities and serve 1.95 B of them.

The assembled model does not beat a dense baseline of comparable size: 57.9% against 82.6% on 178 tool-choice probes, and 36.5% under the contiguous regrouping of Section 11.1. Two defects are identified and fixed in code and have not been re-run, and a third sampling arm is trained and unscored. The architecture is not validated by this paper and we do not claim it is.

The project holds one result that runs the other way, and it is older than this model. At 3.4 M parameters on a retrieval task, a compositional system beat a monolith of matched depth and matched budget, 0.7877 against 0.7350, and the ordering that did it is the ordering XE5 is built in: language first, specialists next, memory stage last, no return path. That settles where the layers go. It is our own experiment at 3.4 M against this model's 127.5 M and on a different task, so it is not the architecture's validation and we do not present it as one. The accurate statement of the project's position is that the architecture is unproven at the XE5 scale, not that it is unproven.

One measurement here is independent of all of that. The cost of omitting a syntactic construction from supervised data varies several-fold between constructions, the ranking is stable at the extremes, and it replicates on a public benchmark with a size-matched control. Coverage is a short list to get right, not a volume to increase.

The thing that would settle the architecture is a benchmark that does not exist yet. Our corpus determines the specialist from the input alone, so a router scores identically on every probe in this paper, and the separating task is memory-contingent: the same utterance requiring a different specialist depending on what is stored. Until that exists, what this paper reports is a system, its measurements, and the ablations that say where the current deficit sits.

Appendix A A state-space alternative to attention

This sits in an appendix rather than in the body because nothing in this paper uses it. The two forms are verified against each other and the mixer has never been trained, so it belongs with the work that is specified and not yet measured rather than among the results.

Softmax attention is expensive because it does precise retrieval over the entire sequence: every query sees every key, which costs $O(n^2)$ to train and a cache that grows with the conversation to serve. In this architecture that is duplicated work. Precise retrieval is what the memory already does, with derived keys, a beam descent and a printable path. The sequence mixer does not also need to be a retrieval engine; it needs to carry local dynamics.

The alternative implemented here is a gated linear recurrence [33, 34] with a per-head decay learned through a sigmoid, so each head chooses its own horizon:

$$S_t = \gamma_h \, S_{t-1} + k_t^\top \, v_t, \quad y_t = q_t \, S_t \tag{14}$$

Two forms of the same function. The recurrent form serves: the state is fixed size, so inference is O(1) per token in both time and memory and there is no cache to grow. The parallel form trains: a decay matrix makes it one masked matmul with no softmax anywhere. The two agree to $\mathbf{2.442\times10^{-15}}$ in double precision over 64 positions, which is the only property that makes the pair usable, because a model trained in one form and served in the other must compute the same function.

Table 9: Inference state at d = 128, 4 heads. The recurrence is constant in context by construction; the attention cache is linear in it. The ratio at 8,192 tokens is 512×.

Context	Recurrent state	Attention cache
128 tokens	16.0 KB	128.0 KB
1,024 tokens	16.0 KB	1,024.0 KB
8,192 tokens	16.0 KB	8,192.0 KB

One implementation detail is worth recording because getting it wrong is silent. Normalisation must be per-head and per-position. A GroupNorm over [B, D, T] normalises across time, which makes the layer's output depend on how many tokens are fed at once; the parallel and recurrent forms then stop being the same function and a model trained one way serves wrong the other. Anything touching the time axis is forbidden inside the mixer.

The cost, stated honestly. Linear attention is weaker than softmax at exact copying, which is why induction heads matter. That is the trade being made deliberately, on the argument that exact recall is delegated to the memory rather than expected from the sequence mixer. **This mixer is not trained.** It is a drop-in for the block with the two forms verified against each other and nothing more; no XE5 checkpoint in this paper uses it, and the argument that the memory can carry the recall the mixer gives up is untested.

References

[1] Hupkes, D., Giulianelli, M., Dankers, V., et al. A taxonomy and review of generalization research in NLP. *Nature Machine Intelligence*, 5:1161–1174, 2023.

[2] Larson, S., Mahendran, A., Peper, J. J., et al. An evaluation dataset for intent classification and out-of-scope prediction. *EMNLP*, 2019.

[3] 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.

[4] Ben Allal, L., Lozhkov, A., Bakouch, E., et al. SmolLM2: when Smol goes big, data-centric training of a small language model. *arXiv:2502.02737*, 2025.

[5] Weber, L., Jumelet, J., Bruni, E., Hupkes, D. Language modelling as a multi-task problem. *EACL*, 2021.

[6] Hierarchical episodic memory with top-down retrieval (ECHO). *arXiv:2605.10993*, 2026.

[7] Nearest-neighbour expert routing. *arXiv:2601.02144*, 2026.

[8] Hierarchical memory over frozen anchor models. *arXiv:2510.02375*, 2025.

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

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

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

[12] De Lange, M., Aljundi, R., Masana, M., et al. A continual learning survey: defying forgetting in classification tasks. *IEEE TPAMI*, 44:3366–3385, 2022.

[13] Hoffmann, J., Borgeaud, S., Mensch, A., et al. Training compute-optimal large language models. *NeurIPS*, 2022.

[14] Shumailov, I., Shumaylov, Z., Zhao, Y., et al. AI models collapse when trained on recursively generated data. *Nature*, 631:755–759, 2024.

[15] Lin, Z., Gou, Z., Gong, Y., et al. Not all tokens are what you need: selective language modelling. *NeurIPS*, 2024.

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

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

[18] Ba, J. L., Kiros, J. R., Hinton, G. E. Layer normalization. *arXiv:1607.06450*, 2016.

[19] Dao, T., Fu, D. Y., Ermon, S., Rudra, A., Ré, C. FlashAttention: fast and memory-efficient exact attention with IO-awareness. *NeurIPS*, 2022.

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

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

[22] Radford, A., Wu, J., Child, R., et al. Language models are unsupervised multitask learners. OpenAI technical report, 2019.

[23] Loshchilov, I., Hutter, F. Decoupled weight decay regularization. *ICLR*, 2019.

[24] Loshchilov, I., Hutter, F. SGDR: stochastic gradient descent with warm restarts. *ICLR*, 2017.

[25] Paszke, A., Gross, S., Massa, F., et al. PyTorch: an imperative style, high-performance deep learning library. *NeurIPS*, 2019.

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

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

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

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

[30] Borgeaud, S., Mensch, A., Hoffmann, J., et al. Improving language models by retrieving from trillions of tokens. *ICML*, 2022.

[31] Wu, Y., Rabe, M. N., Hutchins, D., Szegedy, C. Memorizing Transformers. *ICLR*, 2022.

[32] Su, J., Lu, Y., Pan, S., et al. RoFormer: enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[33] Katharopoulos, A., Vyas, A., Pappas, N., Fleuret, F. Transformers are RNNs: fast autoregressive transformers with linear attention. *ICML*, 2020.

[34] Gu, A., Dao, T. Mamba: linear-time sequence modeling with selective state spaces. *COLM*, 2024.

[35] Rae, J. W., Potapenko, A., Jayakumar, S. M., Lillicrap, T. P. Compressive transformers for long-range sequence modelling. *ICLR*, 2020.

[36] Shazeer, N. Fast transformer decoding: one write-head is all you need. *arXiv:1911.02150*, 2019.