TRAIN SHORT, REASON LONG: COMPOSITIONAL GENERALIZATION VIA FROZEN HOLOGRAPHIC MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Trained sequence models compose known facts at the depths they trained on but degrade beyond them. We trace the failure to one architectural choice: trained weights both decide which facts to compose and execute the composition, so an approximation in one corrupts the other. Systems that keep execution symbolic avoid the choice, but where their rules compose along one path over a fixed set of facts, a fact off the path is never read and a fact that changes is never replaced. We present Frozen Holographic Memory (FHM), an architecture in which trained weights decide what enters a composition and a fixed algebra executes it. One algebra holds facts, replaces them exactly, composes them, and induces rules from them. Trained on possession chains up to two facts deep, FHM tracks state exactly on chains 40 facts deep. Trained on CLUTRR stories of two or three facts, it composes ten-fact stories at 100.0% on the net generation count and 98.3% on the full kinship label. On NoRA, FHM induces its own rules and constraints and scores above the Edge Transformer on every split scored, 52.8% against 6.0% on the split whose answers need the most facts off the path. We intend FHM as an architectural primitive for any system that requires state tracking, composition at unseen depths, rule induction, or off-path reasoning.

1 INTRODUCTION

Consider this question: if Diana is the mother of Ken, and Ken is the father of Ava, how is Diana related to Ava? The "grandmother" fact is never stated, but is derived from composing two facts that are. Answering such questions requires systematic generalization, the ability to recombine known facts and rules into combinations never seen (Fodor & Pylyshyn, 1988; Hupkes et al., 2020). Trained sequence models generalize to inputs of the lengths and depths they trained on but degrade beyond them (Anil et al., 2022; Kim & Linzen, 2020). Language models solve compositional tasks by matching them to linearized subgraphs seen in training, and the match fails once a problem is wider or deeper than any pattern they have seen (Dziri et al., 2023). Under standard conjectures, a log-precision transformer belongs to the circuit class TC^0 (Merrill & Sabharwal, 2023) that cannot compose a sequence of arbitrary length (Merrill et al., 2024).

We trace the failure to one architectural choice: trained weights both decide which facts to compose and execute the composition, so an approximation in one corrupts the other. Systems that keep execution symbolic, such as R5 (Lu et al., 2022), avoid the choice. In R5, a tree search picks which two relations to compose, and a stored rule composes them. After training on CLUTRR stories (Sinha et al., 2019) of up to four facts, R5 answers stories of ten facts. R5 composes relations along one path at a time between the two entities in a question, as do the other path-based methods that do best on CLUTRR (Das et al., 2025). A fact off every such path is never read, and because the story arrives as one graph, a fact that changes is never replaced.

We present Frozen Holographic Memory (FHM), an architecture in which trained weights decide what enters a composition and a fixed algebra (Plate, 1994) executes it. One algebra holds facts, replaces them exactly, composes them, and induces rules from them. The algebra acts on a memory, an array of vectors, and everything the memory holds is filed under a key and read back with that key. A composition is a sequence of reads, and every read returns exactly what was stored within a

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

computed capacity. The trained weights are linear layers, one for each decision a task learns, that learn from answers alone. Every trained decision is discrete, one choice per input, and every read returns a codebook entry.

We evaluate FHM on three tasks. Possession asks who holds a queried item after a chain of transfers, with held-out names. CLUTRR asks how one family member relates to another, in two projections: the signed number of generations between the two, and the full kinship label (Sinha et al., 2019). NoRA asks for every relation that must be true of a queried pair, over stories whose facts follow rules, and we score its four held-out splits that involve ambiguous facts (Das et al., 2025).

Section 2 describes the architecture, Section 3 the experimental protocol, and Section 4 the results.

2 ARCHITECTURE

FHM consumes and reasons over *concepts*. A concept is anything the memory can represent: a person like Diana, or a relation like "mother". A *fact* consists of a subject, a relation, and a value; a set of facts is called a *story*. A *question* is a fact with one part missing, and asks for that part from the contents of a story. Section 1 opens on a story of two facts, Ken : mother : Diana and Ava : father : Ken, and a question, Ava : ? : Diana. Section 2 reuses this story in its examples.

2.1 CODES AND THE ALGEBRA

A *code* is a vector of D complex numbers that represents one concept, and one entry of a code is a *frequency bin*, or simply a *bin*. A code whose magnitudes are 1 at every bin is *unitary*. Fixing every magnitude at 1 leaves each bin with a single differentiator: its angle around the unit circle. That angle is the bin's *phase*, and in the standard notation for complex numbers, bin j is written $e^{i\phi_j}$, where ϕ_j is its phase. A unitary code is therefore fully specified by its D phases. Every drawn code is unitary. The memory draws its codes as random phases and freezes them, meaning their phases never change. The D bins together are a code's *spectrum*.

Plate defined *binding* and *unbinding* (Plate, 1994). Binding associates two concepts. The binding of x and y is their elementwise product $x \odot y$. At every bin, the two phases add and the magnitudes multiply, so the product of unitary codes is itself unitary. The product is associative, so a product of three or more codes is written without brackets. Unbinding is the inverse and retrieves what binding stored. Because every bin of a unitary code has a magnitude of 1, the complex conjugate $\bar{y}$ is an exact multiplicative inverse of y, $(x \odot y) \odot \bar{y} = x$ (Plate, 1994, §3.6.3).

A fact binds its subject, relation, and value codes by elementwise multiplication. To store "Diana is the mother of Ken", we bind the code for *mother-of-Ken* with the code for *Diana*. *Mother-of-Ken* is itself a binding, the relation code for *mother* bound to Ken's code. Unbinding the fact with that binding's inverse recovers Diana's code exactly. The code a read unbinds with is the binding's *key*, and the code the read returns is its *value*. Appendix A visualizes this derivation.

2.2 BUNDLES, CROSSTALK, AND CAPACITY

A memory may hold many bindings at once. *Superposition* combines bindings by using addition. A *bundle* of two bindings is their sum, $t = k_1 \odot v_1 + k_2 \odot v_2$. The sum of two bindings is no longer unitary (Plate, 1994, §4.2), because at each bin, it adds two complex numbers of magnitude 1, whose sum has a magnitude anywhere from 0 to 2. Unbinding k_1 from t gives v_1 plus a second term, $k_2 \odot v_2 \odot \bar{k}_1$, the *crosstalk*, the noise of everything else the bundle holds, turned by k_1's inverse. Crosstalk's correlation with any stored code is the *cosine similarity*, or simply the *cosine*, between the two spectra, accumulated over all D bins. That cosine concentrates near zero, with variance inversely proportional to D (Plate, 1994, Table 4.2). Crosstalk grows with *load*, the number of bindings a bundle holds. A value read from a bundle of m bindings scores a cosine near $1/\sqrt{m}$ against its own code. Additive updates are a second source of crosstalk, because overwriting a value by addition leaves the old binding in the sum.

The classical remedy is Plate's *clean-up memory* (Plate, 1994, §3.3), a codebook that holds every code a read may be replaced by. A *codebook* is a table of codes, one code per concept. A noisy read is replaced by the nearest code in the codebook, or rejected when no code's cosine clears a defined

floor. This paper calls the replacement *cleanup*, or a *snap*. Appendix A carries out a superposition, a bundle read, and the snap on two of the example's facts.

Every read recovers the exact stored code within computed capacity. How much load a bundle tolerates depends on the read that consumes it. An *argmax* read snaps to the best-scoring code, so only the true code needs to outscore every competitor. A *presence* read asks whether any value is present, and compares the best score to a *presence floor*, which a present value must clear. A bundle has two capacities, one per criterion, and the argmax capacity depends on a third quantity besides D and the load: the size of the codebook the read must discriminate over. A bundle's *capacity* for a read is computed from the crosstalk statistics as the largest load at which a read fails its criterion with probability below 10^{-4} (Appendix B.1).

The architecture keeps four invariants, and Section 4 removes them one at a time.

- *Unit-magnitude keys.* The conjugate inverts a key exactly only when the key is unitary.
- *Exact replacement.* Replacing a value subtracts its old binding before adding the new one (Section 2.3).
- *Re-discretization.* Every read returns members of a codebook, or none when no code clears the floor, so no stored value carries another read's crosstalk forward.
- *Capacity matched to the read's criterion.* A bundle is sized to the computed capacity of the read that consumes it.

2.3 The store and its edit

A *store* is an array of bundles. A store files each fact under the fact's subject; a fact whose subject is Ken is filed in a bundle assigned to Ken, for example. A subject's facts may span multiple bundles within a single store. Since phases can add in either order, binding is commutative, meaning a fact cannot distinguish which of its codes is its subject, relation, or value. Therefore, the memory draws two codes for every concept that can be a subject or a value: a *subject code* and a *value code*. A fact's key is its relation code bound with its subject code, built the way *mother-of-Ken* was in Section 2.1 with the subject code in Ken's place. Storing Ken : mother : Diana adds the key bound with Diana's value code to a bundle assigned to Ken.

A store has one operation for writing a fact. Its edit reads a subject's stored value before writing and removes the old value exactly, so a later fact replaces an earlier one. The read unbinds the subject's bundle by the fact's key and scores the result against the value codebook. When the best score clears the presence floor, the subject already holds a value under that key, whose binding the edit subtracts before adding the new one. When no score clears the floor, the edit only adds the new binding. A trained controller scales the whole edit by one number, its *gate* g, which is 1 or 0 at evaluation (Section 2.4), so the edit in full is

$$b' = b + g\left(k \odot v_{\text{new}} - [\text{present}]\, k \odot v_{\text{old}}\right), \tag{1}$$

where b is the subject's bundle before the edit and b' after it, k is the key, v_{new} the value being written, and v_{old} the value the read found, and the bracket is 1 when the best score clears the floor and 0 otherwise. Because every edit begins with the presence read, a store that replaces values packs its bundles to the presence capacity.

With g at 1 or 0, exact subtraction enforces the exact-replacement invariant and gives the store *history independence*, meaning the crosstalk from any read depends only on the facts the store currently holds, so crosstalk does not accumulate with edits.

2.4 The controllers

The edit of Section 2.3 leaves its gate g for a controller to set. The *controllers* are the only part of the architecture with trained weights, one for each kind of decision a task has to learn. A controller reads one input s, a code or a bundle representing what is to be decided, and makes one decision for each input. It chooses among N options that the rest of the architecture then executes. A controller is one linear layer followed by a softmax,

$$f(s) = \frac{[\text{Re } s;\ \text{Im } s]}{\|[\text{Re } s;\ \text{Im } s]\|}, \qquad p = \text{softmax}(W f(s) + \beta)\,, \tag{2}$$

where s is read as its real and imaginary parts and scaled to unit length, and $W \in \mathbb{R}^{N \times 2D}$ and $\beta \in \mathbb{R}^N$ are the trained parameters, $N(2D+1)$ in all. Which input a controller reads is part of each task's wiring (Section 3).

The output p gives each option a probability. For the gate's controller, whose options are to write the fact or to do nothing, g is the entry of p for writing. In training, every option is applied, weighted by its probability. The only loss is on the question's answer, and its gradient reaches every decision through those weights. Evaluation applies only the most probable option, which is a *hard choice*. The hard choice is the re-discretization invariant of Section 2.2 applied to decisions. For the gate, it puts g at 1 or 0, as the history independence of Section 2.3 requires.

2.5 Composition

The question Ava : ? : Diana names two concepts that no fact in the story links. Its answer is composed along the path of facts between them, where each fact's value is the next fact's subject. Along the path, FHM composes both a count and the relation linking the two concepts.

Every relation adds an integer to the count, its *offset* n. In our example, the count follows steps up and down a family tree, so a parent is -1 and a child is $+1$. One code u is drawn, the *step code*, and the code for each integer n is the elementwise power u^n, whose bin j is $e^{in\phi_j}$ (Plate, 1994, §3.6.5). The offset $+1$ is u itself, -1 is u^{-1}, and 0 is u^0. Binding the powers of two offsets adds the offsets,

$$u^{n_1} \odot u^{n_2} = u^{n_1+n_2}, \tag{3}$$

which is the integer case of fractional power encoding (Komer et al., 2019; Frady et al., 2021). A *register* is a code that keeps the running count along a path. It starts as u^0, and each fact binds it with the power of its relation's offset. Because a relation's offset is never stated, the *offset controller*, a controller of Section 2.4, learns it from questions whose answers are known. Its input is a relation's code, and its N options are the offsets. With the learned offsets, the register ends the path from Ava to Diana as $u^0 \odot u^{-1} \odot u^{-1} = u^{-2}$.

A register can be used only when the facts on the path are known, but a story does not say which of its facts those are. For a story, FHM computes the count with a *count store*, a store that holds one count for every concept, each counted from the question's subject. The question's subject, Ava, is given the count u^0. When a fact's subject has a count and its value does not, the store binds that count with the power of the relation's offset and files the result under the value's subject code. The answer is the count that reaches the question's value, Diana, read by a snap whose codebook is the powers of u (Appendix B.2).

The relation that links Ava to Diana is composed from the two relations on the path: father, and then mother. The relation that an ordered pair of relations composes to is the pair's *product*; the product of father, then mother, is grandmother. Products are knowledge about the world, so the algebra cannot compute one, and a *product store* supplies it. The product store files each product under a key built from its pair. Binding is commutative (Section 2.3), so a key made of the two relation codes alone would be the same for father then mother as for mother then father. The pair key $k_{1,2}$ binds each relation code, r_1 and r_2, with a drawn *side code*, ρ_{left} for the first relation and ρ_{right} for the second,

$$k_{1,2} = r_1 \odot \rho_{\text{left}} \odot r_2 \odot \rho_{\text{right}}. \tag{4}$$

The product store is written from questions whose answers are known, because such an answer is the product of the relations on the question's path. A write reads the pair's key first, as the edit of Section 2.3 does, and replaces a different product it finds there with a *conflict marker*,

$$b' = b + [\text{new}]\, k_{1,2} \odot v + [\text{contradicts}]\, \big(k_{1,2} \odot c - k_{1,2} \odot v_{\text{old}}\big), \tag{5}$$

where b is the pair's bundle, v the product's code, v_{old} the code already filed, and c the conflict marker's code. The bracket [new] is 1 when nothing is filed under the key, [contradicts] is 1 when a different product is filed there, and both are 0 when the same product is. A conflict marker is never replaced, so two stories that disagree about a pair leave the marker under its key in either order. Paths of more than two relations contract through the store: a pair of adjacent relations is replaced by its product, and the replacement repeats until one relation is left. Any order of replacements may be used, as long as each pair it replaces has a product filed (Appendix B.3). When every order runs into a pair with no product filed, the product store has no answer, and the task's wiring decides what is answered instead (Section 3.2).

2.6 RULES AND OFF-PATH REASONING

Suppose we add two facts to the story: Cecilia is an aunt of Ava, and Diana has zero daughters. The story does not say which side of Ava's family Cecilia is on. An aunt is a parent's sister, so Cecilia is either Ken's sister or the sister of Ava's mother. If she were Ken's sister, she would be Diana's daughter, but Diana has zero daughters. Cecilia is therefore Ava's maternal aunt. The answer depends on Diana having zero daughters, but Diana is on no path between Cecilia and Ava. Answering from such a fact is *off-path reasoning* (Das et al., 2025).

To reach an off-path fact, a store has to be readable from both ends of every fact. For these questions, a story's facts are filed twice, once under each end. The second filing is bound with a drawn code that marks it as reversed. The aunt fact's subject is Ava, but its reversed filing lets a read from Cecilia return Ava as well. A read from Ava returns Ken, a read from Ken returns Diana, and a read from Diana returns that she has zero daughters.

A *rule* pairs an *if-part*, a few facts over placeholders, with a *then-part*, the fact that follows. An if-part *matches* a story when every one of its facts is read from the story with the placeholders filled by distinct concepts. A product of Section 2.5 is a rule: two facts in a row, and the fact they compose to. A rule's if-part may also name a fact off the path. The reasoning above is one such rule,

$$Y : \text{aunt} : X, \quad Y : \text{grandparent} : Z, \quad Z \text{ has zero daughters} \;\Rightarrow\; Y : \text{maternal aunt} : X. \tag{6}$$

FHM induces such rules from questions whose answers are known. A rule is filed in a *rule store*, with the if-part as the key and each of its then-parts as a value under it. For each question, FHM matches every candidate if-part against the story, with X and Y as the question's two concepts, and files each match with the question's known relations as its then-parts. When an if-part matches a question's story but one of its then-parts is not among that question's known relations, the then-part takes the conflict marker. A then-part that is filed as often as the *filing floor* requires and never takes the marker becomes a rule with its if-part, if applying the rule contradicts no known answer. Appendix B.4 gives the candidate if-parts, the key's construction, the filing floor, and the certification.

Rules are applied in rounds. In each round, every rule is applied wherever its if-part matches, and its then-part is written into the store as a new fact of the story. Rounds repeat until nothing new is written, so a fact written in one round can be read in the next. In the example, a first round composes the path from Ava to Diana, as a product of Section 2.5 does, and a second round applies the rule of Equation 6 to what the first wrote,

$$\begin{aligned}&\text{Ava} : \text{father} : \text{Ken}, \quad \text{Ken} : \text{mother} : \text{Diana} \;\Rightarrow\; \text{Ava} : \text{grandparent} : \text{Diana},\\ &\text{Ava} : \text{aunt} : \text{Cecilia}, \quad \text{Ava} : \text{grandparent} : \text{Diana}, \quad \text{Diana has zero daughters}\\ &\qquad\Rightarrow\; \text{Ava} : \text{maternal aunt} : \text{Cecilia}.\end{aligned}$$

Some stories leave a fact open between two or three values, and a *constraint* rules out the ways of filling it that cannot be true. A constraint is an if-part with no then-part, two facts that cannot both be true. FHM proposes a constraint from counts over the stories whose answers are known, either two facts that never occur together although their counts predict they would, or a relation that never links one concept to two others (Appendix B.4). A proposed constraint is kept only if it matches none of those stories after the rules are applied. FHM answers a story with an open fact once for each way of filling it and drops the ways a constraint matches. The answer is what is true in all the rest.

3 EXPERIMENTAL PROTOCOL

Every task trains FHM on short stories and evaluates it on deeper chains on possession, longer paths on CLUTRR, and harder splits on NoRA. Possession and CLUTRR each train one controller while NoRA trains none. Every code has $D = 256$ bins and the presence floor is 0.5. On possession and CLUTRR, we compare against a transformer trained from scratch on the same stories. On NoRA, we compare against the Edge Transformer (Bergen et al., 2021), the best model NoRA reports.

3.1 POSSESSION

A possession story is a list of events, each naming a person, a verb, and an item, and its question asks who holds one of the items at the end. Every item is introduced first with its holder, one of

four people. One item, the focal item, then passes from person to person, and the number of these transfers is the story's depth. The other items are distractors and stay with their first holder. Half of the stories end with one distractor handed to a different person. Four of the twelve verbs introduce an item, four transfer it, and four only observe it, but a story never says which verb is which. Before each transfer of the focal item, there is a 60% chance of a decoy, an observation event that names an item together with someone who does not hold it.

Names are random strings of three to six letters, drawn once for 12 people and 340 items. Training uses eight of the people and 200 of the items, and evaluation uses the other four people and 140 items. Training stories are one or two transfers deep with two to five distractors. Half of their questions ask about the focal item, and the rest ask about an item picked at random. Evaluation stories form a grid of depths 2, 5, 10, 20, and 40 crossed with story sizes of 5, 20, 60, and 100 items, 300 stories to a cell. Every cell has a twin of the same depth and size drawn from the training names, which separates the effect of depth from the effect of held-out names. Every evaluation question asks about an item picked at random, so the focal item is asked about in only 429 of the 6,000 evaluation stories, and in 2 of the 300 stories in the cell of depth 40 and 100 items.

The transformer reads each story as text, one clause of person, verb, and item per event, ending with the queried item and a question mark. FHM reads each event as a fact whose subject is the item, whose value is the person, and whose relation is the same for every event. The gate's controller reads only the event's verb and decides whether the edit of Section 2.3 writes the event's fact. The verb reaches the controller as its *signature*, a bundle with one binding per letter, each letter's code bound with a code for the letter's position in the word. FHM answers the question with a read at the queried item, which the restricted readout of Section 3.4 snaps to the value codes of the story's four people. The unrestricted readout snaps the read to the value codes of every person and item, and spells that concept's name from a *content store*. The content store files each concept's name one letter per binding, each letter's code bound with a code drawn for the concept and a code for the letter's position. A read under a concept's code and one position code returns that position's letter, so a name is spelled one position at a time. The content store is written once and read only by argmax, so its bundles are filled to the argmax capacity for its letter codebook.

3.2 CLUTRR

CLUTRR is a published benchmark of family-relation questions (Sinha et al., 2019). Each question names two people and asks what relation the second has to the first. We use the released split `gen_train23_test2to10`, which trains on 9,074 stories of two or three facts and evaluates on 1,146 stories of two to ten facts.

We use each story's facts as the benchmark lists them, with its question and answer. In the transformer's text, each person is a lowercase letter, each relation is an uppercase letter, facts are separated by periods, and the question is the two letters of the queried pair. Appendix C.1 shows one story in both forms. Stories use fourteen relations, and answers come from twenty labels: the fourteen plus six that arise only by composition, such as niece and mother-in-law.

The net generation count is the signed number of generations from the first person in the question to the second, and the full label is the relation itself. In the data, every relation has a fixed offset. The labels span up to two generations, so the count is one of five values from -2 to $+2$. FHM answers the count from its count store and the label from its product store (Section 2.5). Where the product store has no answer, FHM answers from a lookup table built from the training split alone, keyed by the path's net generation count and its last relation.

3.3 NoRA

NoRA is a published benchmark of relational reasoning (Das et al., 2025). Questions name two concepts, two people or a person and a place, and ask for every relation between them. Some facts are ambiguous, meaning a relation links a concept to two or three candidates without saying which. A copy of the story with every ambiguous fact settled one way is what NoRA calls a *refinement*, and a refinement that a constraint matches is discarded. A relation is in the answer when it is true in every refinement that remains.

NoRA has four held-out splits. Test-D questions take more rule applications to derive the answer. Test-W questions have more distinct derivations across their refinements. Test-BL questions have derivations with more rule applications per concept, as when a derivation goes back and forth along one path. Test-OPEC questions have derivations with more facts that lie on no path between the two concepts. We do not score NoRA's three other held-out splits, which have no ambiguous facts.

We use the splits the Edge Transformer's authors used: 12,000 of the 17,712 training questions, the first 12,000 of a fixed shuffle, all 1,123 of Test-D, all 6,452 of Test-W, 12,000 of the 18,443 of Test-BL cut the same way, and all 216 of Test-OPEC. FHM's candidate if-parts have at most three facts over the question's two concepts and at most one other, in the shapes Appendix B.4 lists. The filing floor of Section 2.6 is ten, and we also report every result with it at five. NoRA builds each question around one relation of its answer and, for Test-W, calls a question *hard* when that relation rests on an ambiguous fact and, before any refinement is discarded, is not derived in every refinement.

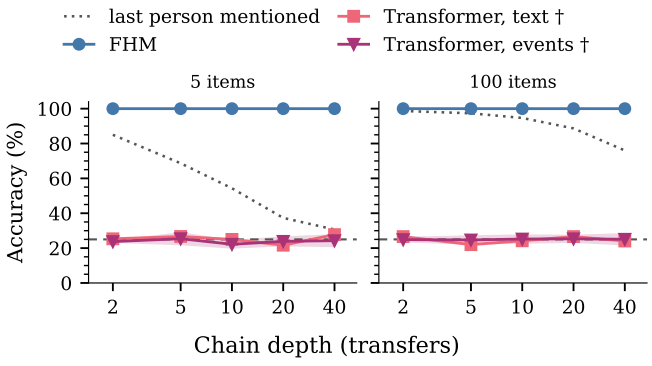
3.4 Arms, training, and metrics

Beside FHM and the transformer, possession has a third arm: the same transformer reading, instead of text, one token per person, verb, and item, and the names of the story's four people. This arm scores state tracking apart from text parsing. The transformer is a six-layer decoder with rotary position embeddings (Vaswani et al., 2017; Su et al., 2024), width 512 and eight heads, with 19.0 million parameters on text and 19.2 million on the event tokens. The transformer's loss is the cross-entropy of the answer tokens, and its learning rate is chosen on validation from four candidates. Each controller trains for 3,000 steps, one story per step, by cross-entropy on the answer at one fixed learning rate. Each controller therefore has 1,026 parameters on possession and 2,565 on CLUTRR. Appendix C.2 lists the training settings. Validation stories on possession have the training names, depths, and sizes. On CLUTRR, they come from the benchmark's validation split. A run has fit its training regime when it answers 95% of them correctly. Possession is scored by two readouts of the answer. The restricted readout chooses the answer among the story's four people, and the unrestricted readout must produce the answer exactly with nothing ruled out, which the transformer decodes and FHM spells. CLUTRR is scored by the restricted readout over its counts or its labels, and NoRA by exact match of the relation set. Possession and CLUTRR run five seeds per configuration.

4 Results

4.1 Possession

Figure 1 shows possession accuracy under the restricted readout against chain depth, one panel per story size. FHM answers every held-out cell at 100.0% under both readouts, in all 20 cells and their 20 twins, across all seeds. The text transformer scores between 20.0% and 28.7% across the held-out cells, and the event transformer between 22.2% and 27.3%. Both score 0.0% under the unrestricted readout. Neither arm fits its training regime on any seed, with validation scoring 81.5% to 88.0% on text and 69.0% to 70.0% on events. On the twin cells with training names at 5 items, the text arm opens at 63.0% at depth 2 and falls to 27.2% at depth 40. On held-out names at the same depth and size, the arm scores 25.3%. The dotted curve scores the guess by recency, the person named in the last event that mentions the queried item. It outscores both arms in every held-out cell.



Restricted readout. Trained at depths 1–2 with at most 5 distractors; held-out names in every cell.
Dashed grey: the four-person chance of 25%. Bands: ±1 s.d. across seeds.

Figure 1: Possession accuracy against chain depth for stories of 5 and 100 items. A dagger marks an arm whose every seed fell below 95% on validation.

4.2 CLUTRR

Figure 2 shows CLUTRR accuracy against story length. On the net generation count, FHM scores 100.0% on all lengths from 2 to 10 on every seed. The transformer scores 100.0% at lengths 2 and 3 and 39.3% at length 10. On the full label, FHM scores 100.0% at length 2 and 98.3% at length 10, and its lowest is 95.2% at length 3. The transformer scores 100.0% at length 2 and 19.5% at length 10. The dotted curve is the share of evaluation stories the product store can answer; 100% at length 2 and 42.9% at length 3. The dip at length 3 comes from 45 stories whose pair, *son* then *father* or *son* then *mother*, composes to the person the story started from, a relation the twenty labels do not include, and 15 whose relations have two answers in the corpus. The product store answers 1,699 of the 2,020 validation stories and is correct on all 1,699. The lookup table answers the other 321 and is correct on 249.

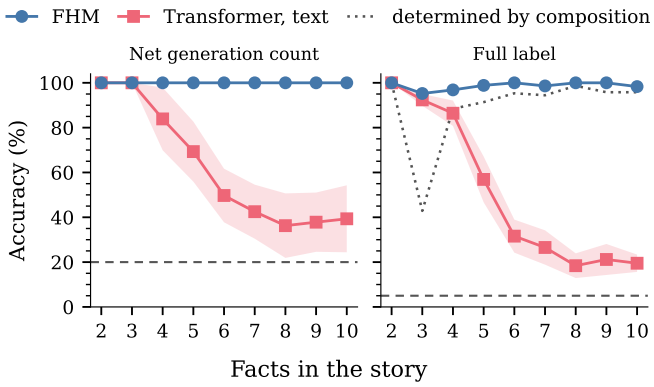


Figure 2: CLUTRR accuracy against story length, the net generation count on the left and the full label on the right.

4.3 NoRA

Table 1 gives NoRA accuracy per split beside the Edge Transformer's two published variants. At a filing floor of ten, FHM induces 4,864 rules and 1,024 constraints from the 12,000 training questions. From these rules, it scores 96.6% on Test-D, 98.1% on Test-W, 98.6% on Test-BL, and 52.8% on Test-OPEC. Of its 430 wrong answers among the 19,791 questions, 425 are missing a relation and 5 have an extra one. At a filing floor of five, FHM induces 7,829 rules and 793 constraints. NoRA marks 390 of Test-W's 6,452 questions as hard. NoRA's definition of hard, applied to the derivation that comes with every released question, marks 389 of the 6,452. On those 389, FHM scores 94.6% at a filing floor of ten and 83.8% at five. The Edge Transformer scores 51% on NoRA's 390.

Table 1: NoRA exact-match accuracy per split, in percent, with the count of hard questions in parentheses. The Edge Transformer's numbers are NoRA's published ones, and NoRA reports one number for its hard questions.

	Edge Transformer		FHM	
Split	Single-edge	Multi-edge	Floor 10	Floor 5
Test-D	74.1	49.3	96.6	94.7
Test-W	70.3	79.0	98.1	90.8
Test-BL	24.5	78.5	98.6	95.7
Test-OPEC	6.0	3.7	52.8	57.9
Test-W, hard	51 (390)		94.6 (389)	83.8 (389)

4.4 Invariant ablations

Table 2 removes the four invariants of Section 2.2 one at a time on possession and reports accuracy over the held-out cells beyond the training depths. The *soft controller* evaluates each decision as its probability-weighted mixture instead of its hard choice. *Capacity mismatch* sizes the store's bundles at the argmax capacity instead of the presence capacity. *Non-unit keys* draws the items' subject codes and the relation code as non-unitary codes instead of unitary ones. The *additive store* drops the subtraction from the edit, so a write adds a new value without removing the old one. The soft controller answers 4,799 of 4,800 stories correctly

Table 2: Possession accuracy over the held-out cells with one invariant removed, mean and standard deviation across seeds, and the seeds that fit their training regime.

Variant	Accuracy (%)	Seeds fit
FHM	100.0 ± 0.0	5/5
Soft controller	99.99 ± 0.01	3/3
Capacity mismatch	88.6 ± 9.7	1/3
Non-unit keys	48.5 ± 17.3	0/3
Additive store	28.2 ± 1.2	0/3

on one seed and all 4,800 on the other two. Capacity mismatch is the one variant whose seeds split, at 100.0%, 88.0%, and 88.5% on the validation stories.

5 Related work

The circuit bound of Section 1 covers state-space models such as Mamba as well (Merrill et al., 2024; Gu & Dao, 2024). Recent fixes keep execution trained and make the trained part more expressive. Allowing negative eigenvalues in the input-dependent state transition lets Mamba and DeltaNet solve parity and improves DeltaNet on harder state-tracking tasks (Grazzi et al., 2025), and PaTH attention applies the same transition inside attention (Yang et al., 2025). The improvement is partial, with DeltaNet extrapolating to about three times its training length on an easier variant of the S_5 permutation task and scoring zero on the full one (Grazzi et al., 2025). In contrast, FHM's state update is not trained (Section 2.3), and its state tracking remains exact at twenty times the training depth (Section 4.1).

R5's stored rules come from a policy-value network and Monte Carlo tree search run during training (Lu et al., 2022). Conditional theorem provers prove a query against rules whose predicates and constants are trained vectors, and learn which rules to apply (Minervini et al., 2020). NCRL trains its rules end to end by merging the relations of a sampled path with a recurrent attention unit (Cheng et al., 2023). R5 and NCRL compose along a path (Das et al., 2025). NoRA's authors found these unsuitable for their benchmark and report as its best model the Edge Transformer, which composes in its weights rather than in stored rules (Das et al., 2025; Bergen et al., 2021). Section 4.3 scores FHM against it. In contrast, FHM's rules are induced from the training questions with no trained part (Section 2.6). Its if-parts may also name a fact off the path, which R5 and NCRL never read.

Overwriting a stored value means removing the old one; memory networks remove it by a graded erase, by an approximate subtraction, or not at all. The Neural Turing Machine and the differentiable neural computer erase a location through a vector of gates between 0 and 1 under a soft weighting over locations, then add the new vector (Graves et al., 2014; 2016). DeltaNet's delta rule subtracts the value read under a learned key and adds a corrected one (Schlag et al., 2021; Yang et al., 2024). Once more associations are stored than the keys have dimensions, that read is a linear combination of stored values, and the subtraction removes the mixture rather than the old value. The classical Hopfield network has no operation that removes a stored pattern (Hopfield, 1982; Krotov & Hopfield, 2016). In contrast, FHM at evaluation fills no bundle past its computed capacity (Section 2.3) and snaps the value it reads to the codebook before subtracting, so what it removes is the stored binding itself (Section 2.3).

The algebra of Section 2.1 is Plate's holographic reduced representation in its complex unitary form (Plate, 1994; 1995), one of the vector symbolic architectures, a family also called hyperdimensional computing (Kanerva, 2009) and surveyed by Kleyko et al. (Kleyko et al., 2022; 2023). Holographic reduced representations have entered deep learning as a differentiable output layer (Ganesan et al., 2021) and as a recasting of self-attention (Alam et al., 2023); in both, the network around the algebra is trained end to end. In contrast, FHM's controllers decide only which facts to write and what each relation adds (Sections 2.4 and 2.5).

6 Conclusion

We presented Frozen Holographic Memory, an architecture whose fixed algebraic operations hold facts, replace them exactly, compose them, and induce rules from them, alongside trained weights that decide what enters a composition. Sections 3 and 4 exercise those operations on state tracking and composition beyond the depths and lengths they trained at, and on rule and constraint induction with off-path reasoning. We intend FHM as an architectural primitive for any system that requires state tracking, composition at unseen depths, rule induction, off-path reasoning, or any combination thereof.

AI USE STATEMENT

We used Claude (Anthropic), through Claude Code, to help validate and carry out our research methodology, write the code that runs the experiments and draws the figures, interpret results, draft and edit parts of this paper, and find related literature. We did not use generative AI to develop the framework, formulate mathematical claims, propose hypotheses, or generate data, and proofs, translation, dataset cleaning, and qualitative analysis do not apply to this work. The architecture, hypotheses, and claims are ours. We reviewed all AI-assisted work, tested the code, and checked every citation against its source, and we take responsibility for the final content, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The code and results are released at `https://anonymous.4open.science/r/trainshort-reasonlong`. Every result in Section 4 is written by the released code into its results files, and Section 4's figures and tables are drawn from those files. The repository's README gives the commands that fetch the two datasets, which it does not redistribute because both are licensed for noncommercial use, and the command that regenerates every results file. Section 3 and Appendix C.2 give the protocol and the training settings, Appendix B.4 the rule store's candidate if-parts, and Appendix B.1 how each bundle's capacity is computed. The possession stories come from a seeded generator and do not need external data. The test suite pins the memory's operations, and the pipeline is deterministic under a fixed Python hash seed.

REFERENCES

Mohammad Mahmudul Alam, Edward Raff, Stella Biderman, Tim Oates, and James Holt. Recasting self-attention with holographic reduced representations. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 490–507. PMLR, 2023. URL https://proceedings.mlr.press/v202/alam23a.html.

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 38546–38556, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/fb7451e43f9c1c35b774bcfad7a5714b-Abstract-Conference.html.

Leon Bergen, Timothy J. O'Donnell, and Dzmitry Bahdanau. Systematic generalization with edge transformers. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pp. 1390–1402, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/0a4dc6dae338c9cb08947c07581f77a2-Abstract.html.

Kewei Cheng, Nesreen K. Ahmed, and Yizhou Sun. Neural compositional rule learning for knowledge graph reasoning. In *International Conference on Learning Representations (ICLR)*, 2023.

Anirban Das, Muhammad Irtaza Khalid, Rafael Peñaloza, and Steven Schockaert. When no paths lead to rome: Benchmarking systematic neural relational reasoning. In *Advances in Neural Information Processing Systems 38, Datasets and Benchmarks Track*, 2025.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaïd Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 70293–70332, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/deb3c28192f979302c157cb653c15e90-Abstract-Conference.html.

Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1–2):3–71, 1988.

E. Paxon Frady, Denis Kleyko, Christopher J. Kymn, Bruno A. Olshausen, and Friedrich T. Sommer. Computing on functions using randomized vector representations. *arXiv preprint arXiv:2109.03429*, 2021.

Ashwinkumar Ganesan, Hang Gao, Sunil Gandhi, Edward Raff, Tim Oates, James Holt, and Mark McLean. Learning with holographic reduced representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 25606–25620, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/d71dd235287466052f1630f31bde7932-Abstract.html.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626): 471–476, 2016. doi: 10.1038/nature20101.

Riccardo Grazzi, Julien Siems, Arber Zela, Jörg K. H. Franke, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear RNNs through negative eigenvalues. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=UvTo3tVBk2.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. doi: 10.1073/pnas.79.8.2554.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020. doi: 10.1613/jair.1.11674.

Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, 2009. doi: 10.1007/s12559-009-9009-8.

Najoung Kim and Tal Linzen. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9087–9105, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.731.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Denis Kleyko, Dmitri A. Rachkovskij, Evgeny Osipov, and Abbas Rahimi. A survey on hyperdimensional computing aka vector symbolic architectures, part I: Models and data transformations. *ACM Computing Surveys*, 55(6):130:1–130:40, 2022. doi: 10.1145/3538531.

Denis Kleyko, Dmitri A. Rachkovskij, Evgeny Osipov, and Abbas Rahimi. A survey on hyperdimensional computing aka vector symbolic architectures, part II: Applications, cognitive models, and challenges. *ACM Computing Surveys*, 55(9):175:1–175:52, 2023. doi: 10.1145/3558000.

Brent Komer, Terrence C. Stewart, Aaron R. Voelker, and Chris Eliasmith. A neural representation of continuous space using fractional binding. In *41st Annual Meeting of the Cognitive Science Society*, pp. 2038–2043, Montreal, QC, 2019. Cognitive Science Society.

Dmitry Krotov and John J. Hopfield. Dense associative memory for pattern recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pp. 1172–1180, 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/hash/eaae339c4d89fc102edd9dbdb6a28915-Abstract.html.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Shengyao Lu, Bang Liu, Keith G. Mills, Shangling Jui, and Di Niu. R5: Rule discovery with reinforced and recurrent relational reasoning. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=2eXhNpHeW6E.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023. doi: 10.1162/tacl_a_00562.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 35492–35506. PMLR, 2024. URL https://proceedings.mlr.press/v235/merrill24a.html.

Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6938–6949. PMLR, 2020.

Tony A. Plate. *Distributed Representations and Nested Compositional Structure*. PhD thesis, University of Toronto, 1994.

Tony A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3): 623–641, 1995. doi: 10.1109/72.377968.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366. PMLR, 2021. URL https://proceedings.mlr.press/v139/schlag21a.html.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L. Hamilton. CLUTRR: A diagnostic benchmark for inductive reasoning from text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4506–4515, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1458.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063. Equation references follow arXiv:2104.09864v5.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 115491–115522, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/d13a3eae72366e61dfdc7eea82eeb685-Abstract-Conference.html.

Songlin Yang, Yikang Shen, Kaiyue Wen, Shawn Tan, Mayank Mishra, Liliang Ren, Rameswar Panda, and Yoon Kim. PaTH attention: Position encoding via accumulating Householder transformations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/59c27bf8d56d3d50c7aeaf7535dee975-Abstract-Conference.html.

A WORKED EXAMPLES

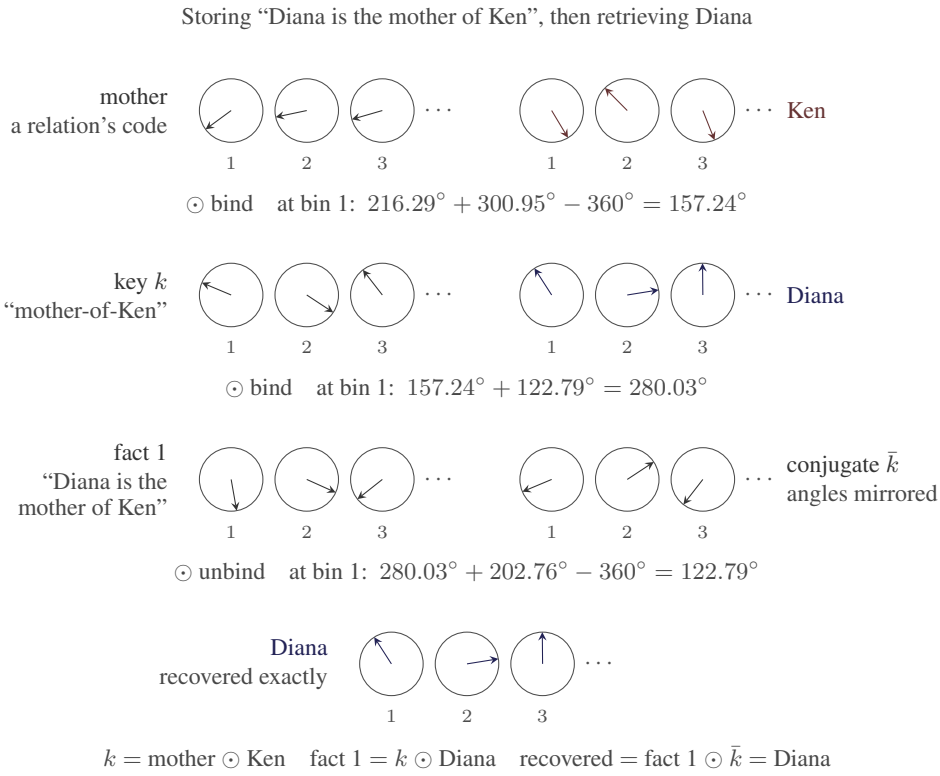


Figure 3: Storing and retrieving one fact on two of the drawn codes and a relation code. Each circle is one bin of a code, drawn as a unit circle whose edge marks magnitude 1, and its arrow points at that bin’s phase. A row draws the first three of a code’s 256 bins, numbered beneath. Each of the first three lines pairs one operation’s two operands, and the operation’s result opens the next line. The angle arithmetic beside each operation is worked to two decimals. Recovered exactly means equal to Diana’s drawn code to floating-point precision.

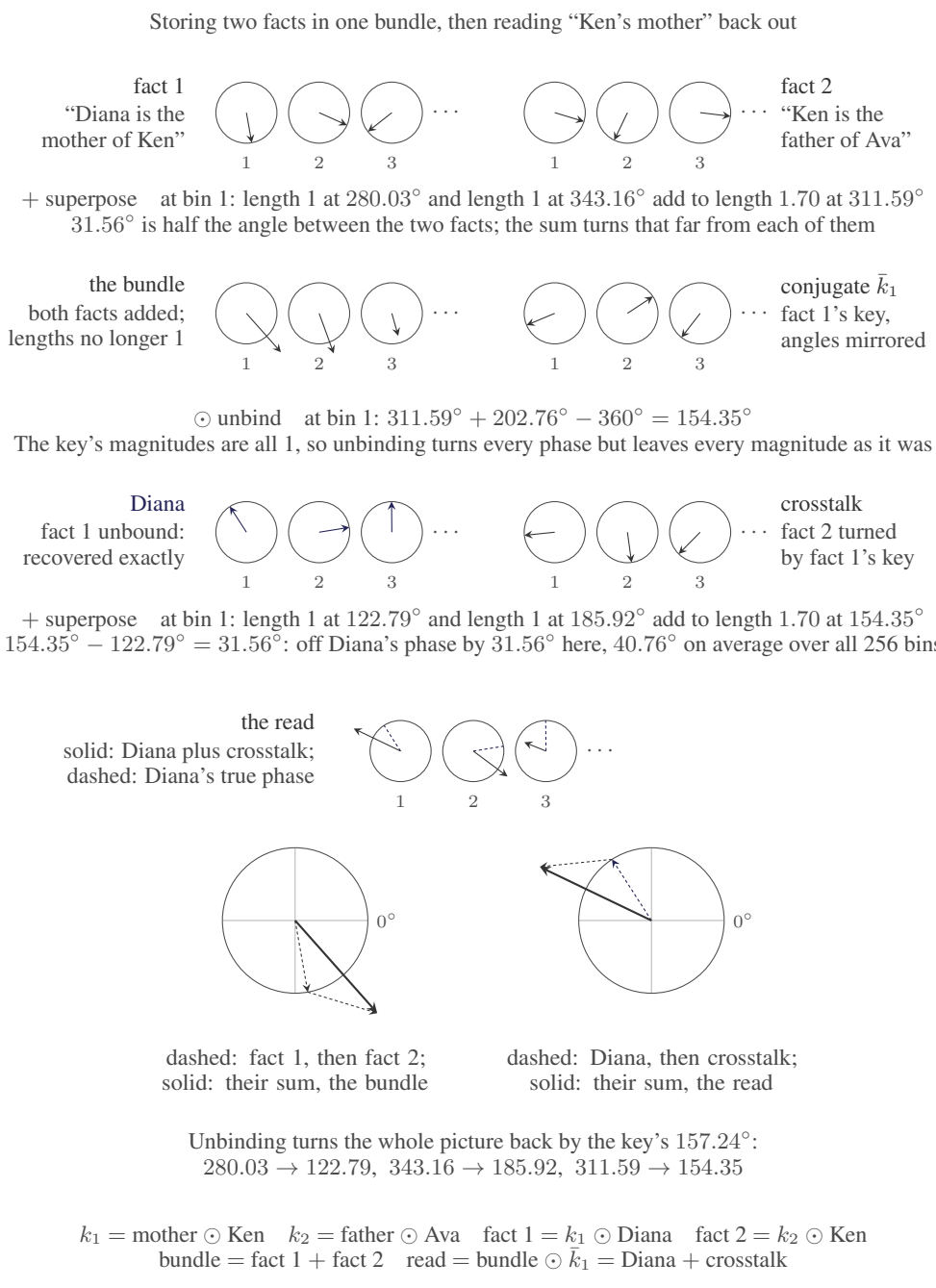


Figure 4: Storing two facts in one bundle and reading "Ken's mother" back, on the codes of Figure 3. Arrow length represents the bin's magnitude. Each row pairs one operation's operands, and the two magnified circles draw bin 1's additions end to end.

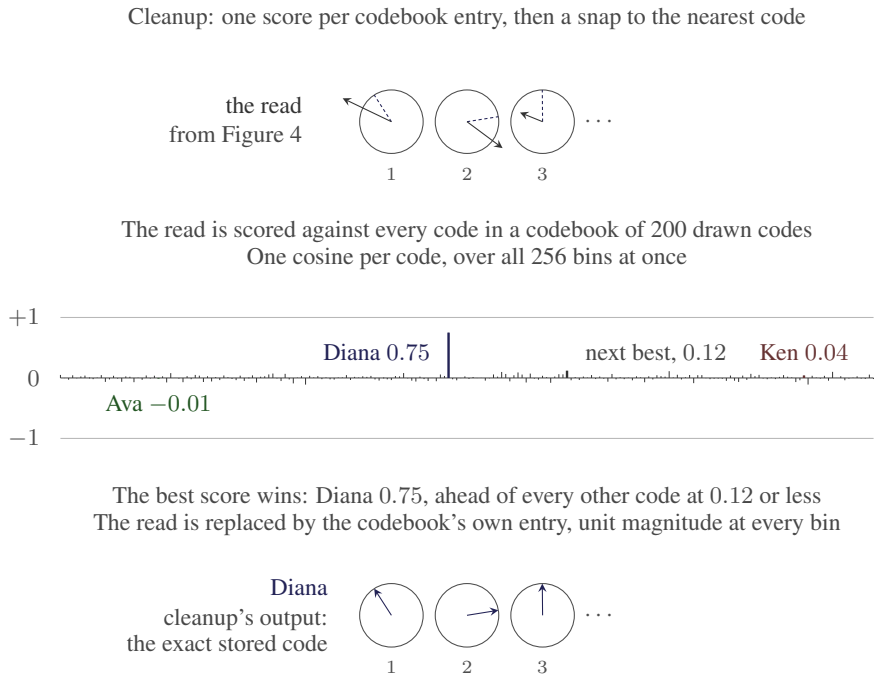


Figure 5: Cleanup of the read from Figure 4, with the scores in codebook order. The same read from a bundle holding fact 1 alone scores 1.00 (Figure 3), so the drop to 0.75 measures the crosstalk the second fact adds. Cleanup's output is Diana's code, restored exactly.

B ARCHITECTURE DETAILS

B.1 CAPACITY FROM THE CROSSTALK STATISTICS

A read of a stored value from a bundle of m bindings is that value plus the $m-1$ other bindings turned by the key. Against its own code the read scores a cosine of mean $1/\sqrt{m}$. The crosstalk's projection onto that code moves the cosine by a standard deviation of $((m-1)/m)^{3/2}/\sqrt{2D}$, and the crosstalk's own norm by $\sqrt{(m-1)(m-2)}/(2m^{3/2}\sqrt{D})$, added in quadrature. Against any other code of the codebook the read scores a cosine of mean zero with standard deviation $1/\sqrt{2D}$ (Plate, 1994, Table 4.2).

A presence read fails when its own-code cosine falls below the floor, and its failure probability at load m follows from the first two statistics. An argmax read over K codes fails when any of the other $K-1$ codes scores above its own. The probability that all stay below a signal value x is $\Phi(x/\sigma)^{K-1}$, where Φ is the standard normal distribution function and σ is the spurious standard deviation, integrated over the signal's own fluctuation. The capacity for a read is the largest load at which its failure probability is below 10^{-4}.

The release code keeps the probes of the earlier draft as tests. A probe loads bundles with fresh random codes until a read fails. At $D = 256$ the computed and probed capacities agree on every codebook the paper uses (Table 3).

Table 3: Computed and probed bundle capacities at $D = 256$ for every codebook the paper uses, with the presence floor at 0.5.

codebook	criterion	codes	computed	probed
possession values	presence	12	2	2
CLUTRR counts	presence	9	2	2
CLUTRR labels and the conflict marker	presence	21	2	2
NoRA concepts	presence	1,024	2	2
possession characters	argmax	128	12	12

B.2 THE PROPAGATION SCAN

The count store of Section 2.5 is filled by a scan. The question's subject is written as u^0 under its subject code, and a scan then visits every fact in the order the story states them. A fact whose subject has a count snaps that count to the nearest power and binds it with the power for the relation's offset, then files the result under its value's subject code unless the value already has a count. After the last scan, the question's value is read once and snapped.

B.3 THE CONTRACTION

Each read of the product store reports its own reliability, the cosine between the read and the snapped code, and the cosine is 0 where the conflict marker wins. A path of relations contracts pairwise through the store, each span resolving through the split whose weakest step is most confident, and a contraction whose best split runs through a conflicted pair has a confidence of 0, so the product store has no answer.

B.4 THE RULE STORE

A rule's then-part is of one of three kinds: a relation between the question's two concepts, X and Y; a *condition*, a fact about one concept X, such as being female or having zero daughters; or a relation between X and itself. An if-part has at most three facts, conditions included, over X, Y, and at most one more concept, Z, and facts may run in either direction. For a then-part between X and Y, the candidate if-parts take four shapes:

- a fact between X and Y, with up to two conditions on X or Y;

- a fact between X and Y, a fact between X or Y and Z, and a condition on Z;
- a fact between X or Y and Z, with one or two conditions on X or Y, at least one on the concept that fact does not name;
- a fact between X and Z and a fact between Y and Z, with up to one condition.

The maternal-aunt rule of Section 2.6 has the second shape. For the other two kinds of then-part, an if-part is one or two conditions on X, or a fact between X and Z with up to two conditions on Z. We chose these shapes from the forms of the rules the benchmark's authors released with it, which FHM never reads.

In the store, a condition is filed as a fact whose relation and value are both the condition's code, so "Diana has zero daughters" is filed as Diana : zero daughters : zero daughters. A read of Diana under that relation returns the condition's code exactly when the condition is true. Each placeholder has a drawn code. A fact's code binds its relation code with the code of its first concept and the square of the code of its second, so a fact and its reverse have different codes, and a condition binds its relation code with its one concept's code. An if-part's key binds its facts' codes with a code τ for the kind of then-part,

$$k = \tau \odot \prod_i r_i \odot a_i \odot b_i^2, \tag{7}$$

where r_i is the ith fact's relation code, a_i and b_i are the codes of its first and second concept, and b_i is the all-ones code when the fact has no second concept.

Under a key, the rule store keeps one binding for each then-part ever filed there, and its bundles spill into new ones at the presence capacity. A read checks each then-part filed under a key for presence and for the conflict marker. Beside its bundles, the store keeps two counts for each then-part under each key: how many filings named it, and how many filings knew its truth. A question about two concepts knows every relation between them. For conditions, FHM first keeps the constraints of the second and third forms below that match no story. A concept knows a condition it states when such a constraint names the condition, and knows a condition is false when such a constraint rules it out. The *filing floor* is the first count a then-part must reach. When the second count exceeds the first, a filing that knew the then-part found it false, and the store files the then-part bound with the conflict marker, subtracting its plain binding if it had one. The counts settle before any edit, so the result does not depend on the order in which stories are filed.

The then-parts at the floor and never marked are candidate rules, unless one repeats a fact of its own if-part, or an accepted rule with the same then-part has an if-part made of some of its facts. The candidates are applied together, with the rules already accepted, to the stories of the questions whose answers are known. A candidate that derives a relation outside a known answer with no other candidate in the derivation is refused. When none is refused that way, the candidate refused is the one that is most often the least-filed step of a contradicted derivation. The refusals repeat until no derived relation is contradicted. Induction runs in up to eight rounds, each filing with the accepted rules already applied, since their facts let more if-parts match. A refused candidate stays refused, and an accepted rule that a later filing marks is refused too.

A proposed constraint takes one of four forms: two relations that never occur in the same known answer; a condition on one of a question's two concepts and a relation that never occur together in its known answer; two conditions that no concept states together; and a relation that never links one concept to two others, or two concepts to one, across 20 or more concepts. Each of the first three is proposed only where the counts of its two parts predict one or more occurrences together.

C EXPERIMENTAL DETAILS

C.1 ONE CLUTRR STORY

The story below is the first in CLUTRR's training split to have two facts. FHM reads its two facts, Donald : father : Michael and Michael : sister : Dorothy, and its question, Donald : ? : Dorothy. The answer is aunt, and its count is -1, since a father is one generation up and a sister is none.

In the transformer's text, Donald, Michael, and Dorothy become a, b, and c, in the order the benchmark numbers them. A relation's letter is its place in the alphabetical list of the fourteen relations: aunt, brother, daughter, father, granddaughter, grandfather, grandmother, grandson, husband, mother, sister, son, uncle, and wife. Father is D, and sister is K. The answer is one character for the count, V to Z for -2 to $+2$, and the label itself for the full label. The transformer reads and writes the text one character at a time. Its restricted readout picks the candidate answer it scores as the most likely completion of the text.

facts	Donald : father : Michael, Michael : sister : Dorothy
question	Donald : ? : Dorothy
answer	aunt, count -1
text for the count	`a D b . b K c . a c ? W`
text for the label	`a D b . b K c . a c ? aunt`

C.2 TRAINING SETTINGS

Each controller trains with Adam (Kingma & Ba, 2015), at a learning rate of 5×10^{-3} on possession and 3×10^{-3} on CLUTRR. FHM scores its validation stories once, after training: 200 on possession and the whole split on CLUTRR. On possession, its validation questions ask about an item picked at random. The controller's scaling of its input to unit length makes selection depend on the input's direction alone. Without it, the softmax would saturate on the input's scale at initialization instead of starting near uniform.

The transformer trains with AdamW (Loshchilov & Hutter, 2019) at $\beta_1 = 0.9$ and $\beta_2 = 0.95$, on batches of 128 stories. Weight decay of 0.1 applies to its parameter matrices only, and gradients are clipped to a global norm of 1.0. The learning rate rises linearly over the first 100 steps and then decays along a cosine to zero at the end of the run's budget.

The learning rate is chosen in two stages. First, one seed trains at each of the four candidate rates, 10^{-4}, 3×10^{-4}, 10^{-3}, and 3×10^{-3}, within a shorter selection budget, and the rate with the best validation accuracy is kept, the smallest on a tie. The final runs then train at that rate within the full budget. Every 250 steps a run scores 200 validation stories. On possession, they are drawn as training stories are. On CLUTRR, they are the first 200 of the benchmark's validation split. A run stops once its accuracy has not improved for a set number of steps, its patience, and it keeps the parameters that scored best. On CLUTRR, every final run scored 100% on validation at its first check, at step 250. Table 4 lists the budgets, the patience, the chosen rates, and the steps the final runs trained.

Table 4: The transformer's step budgets, patience, chosen learning rate, and steps trained across the five final runs of each configuration.

configuration	budget	selection budget	patience	rate	steps trained
possession, text	20,000	8,000	2,000	10^{-4}	6,750 to 10,000
possession, events	20,000	8,000	2,000	3×10^{-4}	2,750 to 6,000
CLUTRR, count	3,000	1,500	500	10^{-4}	750
CLUTRR, label	3,000	1,500	500	10^{-4}	750

C.3 COMPUTE

Every run used one machine with an 8-core CPU and one NVIDIA RTX 5080 GPU. Each FHM run trains and evaluates on the CPU in under a minute, and each transformer run trains on the GPU in under 25 minutes. On the CPU, the NoRA pipeline induces its rules and constraints in about 10 minutes at a filing floor of ten and about 90 minutes at five, and scores the four splits in about 2 hours at either floor.