Inductive Biases for Efficient Next-Token Prediction: Episodic Memory, a Self-Learned Gate, and an Instruction-Copy Path

Brian Musuku

Abstract

A plain autoregressive language model must learn everything—grammar, copying, and unstated facts—through the same next-token objective and the same weight updates. We measure where this is slow or impossible on a controlled instruction-to-code suite whose every target token is labeled at generation time by its source: reusable grammar (T), values stated in the current instruction (S), and house-style constants never stated anywhere (C). Plain training acquires lookup content fast but stated-value copying only after a long plateau, learns the facts of data-scarce families last, and produces values that appear only at test time in 4% of positions—it memorizes context-keyed lookups but builds no copying operation. We then retrain the *same* trunk from the same initialization budget with three inductive biases present from the start and trained jointly from the model's own loss: an episodic store rewritten each epoch from the model's persistent failures, a per-position gate that blends the store with the trunk, and a pointer copy path over the input context blended through a second gate. Nothing outside the loss supervises any of it. The modified model learns dramatically faster and generalizes further: it exceeds the plain trunk's *converged* stated-value copying within 4 epochs, produces novel test-time values at 91% by epoch 4 (the plain trunk never exceeds 4% at any epoch), and recovers data-scarce families' conventions early, while the gates—which never see token roles—divide the work between recalling, copying, and predicting on their own.

1 Introduction

Next-token prediction trains one network, with one objective and one set of weight updates, to do several very different things. Some of what it predicts is *rule-like*: grammar, argument structure, the shape of a correct program—skills a generic sequence model can acquire from a modest number of examples. Some of it is *copying*: reproducing a value that appears in the current input, which is an operation over the context rather than knowledge in the weights. And some of it is *fact-like*: particular constants and house styles that cannot be derived from anything visible and must be stored. A single next-token objective provides no architectural means to distinguish them; everything is left to the same gradient updates, and the resulting model is slow where the task demands copying, slow where data is scarce, and structurally unable to emit a value it was never trained to emit.

This paper makes that diagnosis quantitative and then tests a minimal fix. We build a controlled suite of instruction-to-code pairs whose every target token is labeled at generation time with its source. Each example asks for a small Python function, for instance:

```
Implement scale_amt:  multiply the input by 4 and round the result.
def scale_amt(x):  """scale_amt:  multiply the input by 4 and round."""  x =
float(x); return round(4 * x, 3)
```

Template tokens (`def`, parentheses, the docstring scaffold) are fixed grammar shared across a family of related functions. Slot tokens repeat values stated in the instruction: the function name and the multiplier

4 must be reproduced in the code. Convention tokens are choices that no instruction ever states: the code always begins with `x = float(x)`, and although the instruction says "round the result," it never says to how many decimals—the trailing 3 in `round(4 * x, 3)` is the family's house style, learnable only from other examples of the same family. The corpus deliberately makes the three roles collide—families share convention shapes with different constants (`1e-9` vs. `1e-3` guards, `1.0` vs. `0.5` sentinels)—so that misattributed facts are measurable, and it arranges families into rarity tiers (700, 300, and 70 training examples) so that data scarcity is a controlled variable rather than an accident.

We first train a plain autoregressive transformer and find that its inefficiencies are exactly the ones the token-level ground truth can name. Unstated conventions, which are pure lookup content, are acquired quickly (84.2% by epoch 4). Stated-value copying—an operation, not a lookup—stays below 17% for ten epochs and reaches 95.1% only at epoch 24. Copying of values that appear *only at test time* never exceeds 4%: masking the instruction collapses slot accuracy from 93.5% to 5.3%, showing the trunk uses the instruction as a retrieval key to select among memorized mappings, not as a source to copy from. And the facts of data-scarce families are learned last (33% at epoch 4 for families with 70 examples).

We then ask whether these are failures of inductive bias, not of information, by retraining the same trunk from the same training budget with three biases built in from initialization, all trained jointly from the model's own loss:

1. **An episodic store**, refreshed once per epoch from the model's *persistent prediction failures* (key: trunk hidden state; value: correct next token), starting empty and always tracking what the model still cannot predict—a content-addressable "look it up" pathway.
2. **A self-learned gate**, a small per-position module trained only from loss, deciding how much to trust the store against the trunk.
3. **An instruction-copy path**, a pointer head attending over the instruction tokens, blended through a second gate, giving the model a direct "read it off the context" pathway for stated values.

Nothing outside the next-token loss supervises any of this: the gates never see the token roles, and the store's write criterion reads only the model's own failures. The comparison is deliberately minimal—same architecture for the trunk, same data, same optimizer, same batch order, same number of epochs; the only difference is the presence of the biases and the small modules that operate them.

The result is a large change in what training makes easy to learn (Figure 1). CAMERA—the modified model—exceeds the plain trunk's *converged* stated-value copying within 4 epochs, versus 24 for the baseline; it produces novel test-time values at 91% by epoch 4, a capability the plain trunk never acquires (at most 4% at any epoch); and it recovers data-scarce families early rather than last. The gates, which never see a label, divide the work on their own: the copy gate owns stated-value positions, the store gate concentrates where the trunk still fails, and both release positions as the trunk learns them. The store's composition tracks the model's failures rather than the data's frequencies, which is exactly what the loss-based write criterion predicts.

Three contributions organize the paper: (i) a token-level attribution suite that turns "the model is slow to learn X" into a measurement; (ii) a controlled from-scratch comparison showing that the inefficiencies of plain next-token training are failures of inductive bias—the needed information is present in the data, and a tiny amount of extra machinery makes it learnable many epochs earlier; (iii) evidence that the routing between "recall it," "copy it," and "predict it" can be learned from the loss alone, per position, with no supervision.

2 Related Work

Memory-augmented networks. Equipping networks with an external, content-addressable memory was proposed precisely as an inductive bias for learning algorithms that gradient descent finds hard (Graves et al., 2014; 2016; Santoro et al., 2016). Our store follows this tradition with two differences that matter for attribution: it is written from the model's own training failures (not from end-to-end learned read/write

Table 1: Corpus statistics. Token roles are assigned at generation time: template (T) = fixed grammar shared by a family; slot (S) = value stated in the instruction; convention (C) = constant never stated anywhere. Novel slot values appear in test instructions but never in any training target.

	value
families	12
train examples	4510
train per family (tiers)	700 / 300 / 70
val / test-seen / test-novel	250 / 500 / 250
vocabulary (word level)	335
token role share: T	74.8%
token role share: S	6.1%
token role share: C	19.1%

heads), and its contents are refreshed as training proceeds, so we can observe the memory tracking what the model has and has not learned.

Retrieval and episodic language models. Retrieval-augmented and nearest-neighbor language models move factual load out of the weights at inference time (Khandelwal et al., 2020; Lewis et al., 2020; Borgeaud et al., 2022), and Wu et al. (2023) scale a kNN episodic memory inside a trained transformer. Model-editing systems patch specific facts (Meng et al., 2022) or add episodic stores with learned controllers (Mitchell et al., 2022b;a). Our store differs in what is stored and how its use is learned: entries are *token-level corrective values* keyed on activations, rewritten each epoch from training failures alone, and a per-position gate trained purely from next-token loss—never from any labeling of what should be memorized—decides the blend at each position.

Copy mechanisms. Pointer networks (Vinyals et al., 2015) and sequence-to-sequence copy mechanisms (Gu et al., 2016; See et al., 2017) copy source tokens through attention, and induction heads provide a mechanistic account of copying inside transformers (Olsson et al., 2022). We use a pointer head in the same spirit, but as a training-time inductive bias rather than an inference-time add-on, and in a controlled decomposition: token roles are held out from every module, so the pointer's contribution can be attributed to the bias rather than to supervision.

Delayed generalization. Grokking—long stagnation followed by sudden generalization—was first reported on small algorithmic datasets (Power et al., 2022) and mechanistically traced to the slow formation of generalizing circuits (Nanda et al., 2023). Our slot-copying curves show the same phenomenology in the plain baseline, on a corpus where we can name exactly which capability arrives late: copying stated values into code. The copy-path experiments make the alternative explicit: the same capability, provided as a bias, works from the first epochs of training.

Memorization and capacity. Gradient-based training memorizes examples and noise even at scale (Zhang et al., 2017; Keskar et al., 2019), theory argues long-tailed data is hard to learn *without* memorization (Feldman, 2020), large models memorize far more than they overfit (Tirumala et al., 2022), and capacity analyses model knowledge as a fixed budget of bits (Allen-Zhu & Li, 2024). Our suite makes the contents of parametric memory measurable token by token, and shows that under a fixed budget, memorizing lookups wins the race against acquiring a generalizable copying operation—unless copying is supplied as a bias.

Data distribution and in-context learning. Distributional properties such as burstiness and skew drive the emergence of in-context learning (Chan et al., 2022; Raventós et al., 2023). Our corpus isolates a complementary axis: within a single task, which positions require context reading at all, and what changes when a direct pathway for it is present throughout training.

3 A Task Suite with Token-Level Ground Truth

The unit of data is a pair: a natural-language instruction and a small Python function. Each instruction is drawn from a *family*—a template family such as "multiply the input by s and round the result"—and the target implements it in a fixed house style. Because the corpus is produced by a generator rather than collected, every target token is labeled at construction time with one of three roles (Table 1):

Template (T). Grammar and boilerplate shared across a family: `def`, `return`, `(x):`, the docstring scaffold. Predictable from family membership and syntax alone.

Slot (S). A value that appears verbatim in the instruction and must be reproduced in the code: the function name in the signature and docstring, the multiplier 4 inside `round(4 * x, 3)`. Getting S right requires copying from the instruction context.

Convention (C). A constant choice that appears in *no* instruction anywhere in the corpus: the `x = float(x)` cast, the rounding precision 3, a guard threshold, a fallback sentinel. The only channel that carries C tokens is other training examples of the same family; getting C right requires storing the family's house style.

Engineered interference. Families share near-identical convention shapes with different constants: `if v < 1e-3:` in one family versus `if b < 1e-9:` in another; `return 0.5` versus `return 1.0`. Several families have two code variants selected by instruction phrasing, and families carry group-level trailer comments (`# core`, `# ext`, `# aux`) so that conventions exist at two granularities. When a model emits family A's guard constant inside family B's guard, we count a *contamination error*—direct evidence that a memorized constant was retrieved under the wrong key.

Splits. Per family: training examples, a validation split, a seen-value test split, and a novel-value test split whose instructions contain slot values from a held-out pool (e.g., "multiply by 11" when training held only 2–10 and 12). Novel values *are* in the vocabulary—they occur in test instructions—but never occur in any training target, so producing one cannot be solved by remembering a token; it requires an operation. Rarity tiers (4 families at 700 examples, 5 at 300, 3 at 70) make data scarcity a controlled variable.

4 Three Inductive Biases, Trained Jointly from Initialization

The trunk is a small autoregressive transformer (d_{model}=96, 2 layers, 3 heads, d_{ff}=192, dropout 0.1) over a closed word-level vocabulary. Each example is the sequence [`BOS`] instruction [`SEP`] code [`EOS`], with cross-entropy applied to code positions only. Nothing in the input involves the token roles. We compare two systems trained *from scratch* under identical conditions—same data, same batch order, same optimizer and learning rate, same number of epochs: the plain trunk (AR), and the same trunk with three biases attached from initialization (CAMERA). At initialization the added machinery consists of two small gates (hidden width 64), one linear attention head, and an empty store; all of it trains jointly with the trunk from the next-token loss of the merged distribution.

4.1 Bias 1: an episodic store rewritten from the model's failures

The store starts empty. At the end of every training epoch, we make one teacher-forced pass over the training corpus and record, at every code-predicting position t, the model's loss ℓ_t and hidden state h_t. Per family f with median loss m_f, positions with $\ell_t > m_f$—the *persistent failures*—become store entries, subject to a capacity cap of 20,000. An entry is a key–value pair

$$k_t = \frac{h_t}{\|h_t\|}, \qquad v_t = y_t^\star, \tag{1}$$

where h_t summarizes the instruction and the code generated so far, and $y_t^\star$ is the correct next token. The store is then used as a fixed lookup table for the entire next epoch and rewritten again at its end: the memory always reflects what the *current* model still fails at, not a frozen snapshot of early training. The write criterion is deliberately label-blind: it never consults the S/C roles. Because slots and conventions

are exactly what an under-trained model gets wrong, the criterion *correlates* with those classes—but only through the loss signal, a correlation we verify directly in Section 5.4.

At training and evaluation time, the store retrieves the top K=8 entries by key similarity and turns them into a categorical suggestion

$$p_{\text{store}}(y \mid c_t) = \sum_{i \in \text{top}K} \text{softmax}\big(\beta \,\text{sim}(k_t, k_i)\big)_i \; \mathbf{1}[v_i = y]\,, \tag{2}$$

with temperature $\beta = 10$. Retrieval queries are detached, so the store acts as a lookup table; gradients flow only through the gates and the trunk.

4.2 Bias 2: a self-learned gate for routing

A small gate (hidden width 64) maps four model-derived signals at position t—the hidden state, the maximum probability, the entropy of the model's distribution, and the top-1 key similarity—to a scalar $\alpha_t \in [0, 1]$. The store mixture is

$$p_{\text{mem}}(y \mid c_t) = (1 - \alpha_t)\, p_{\text{trunk}}(y \mid c_t) + \alpha_t\, p_{\text{store}}(y \mid c_t). \tag{3}$$

The bias says: routing should be *per position* and *learned*, not global and hand-tuned. The gate trains jointly with everything else from the loss defined below, plus a penalty that discourages average α above a budget of 0.02; its only supervision is the model's own loss. The token roles, family identities, and store entry labels are never exposed to it.

4.3 Bias 3: a direct pathway from positions to context tokens

The store holds training-time values only, so nothing in Equation (3) can emit a value that appears in no training target. The copy path supplies the missing operation: a pointer over the instruction tokens, in the spirit of pointer networks and pointer–generator hybrids (Vinyals et al., 2015; See et al., 2017). A single attention head computes

$$a_{tj} = \text{softmax}_{j \in \mathcal{I}}\left(\frac{\langle W_q h_t,\, W_k h_j \rangle}{\sqrt{d}}\right), \qquad p_{\text{copy}}(y \mid c_t) = \sum_{j \in \mathcal{I}:\, w_j = y} a_{tj}, \tag{4}$$

where $\mathcal{I}$ is the set of instruction positions in the current sequence, w_j the token at position j, and h_t, h_j the trunk's hidden states. The query reads the prediction position; the keys read the instruction. Repeated instruction tokens accumulate their attention mass, so p_{copy} is a proper distribution over exactly the tokens the instruction contains—and over nothing else.

The final predictive distribution adds a second gate $\gamma_t \in [0, 1]$ of the same architecture, sharing its four inputs:

$$p(y \mid c_t) = (1 - \gamma_t)\, p_{\text{mem}}(y \mid c_t) + \gamma_t\, p_{\text{copy}}(y \mid c_t). \tag{5}$$

Both gates, the pointer, and the trunk train jointly from the next-token loss of Equation (5), plus the episodic budget penalty on α; no penalty is placed on γ. Like the rest of CAMERA, the copy path is label-blind: nothing tells it which positions are slots, or that the name in the signature matches the name in the instruction. Under the instruction-conditioning ablation the instruction positions are masked, no valid keys remain, and γ is forced to 0 by construction—the model then reduces to its store/trunk mixture.

Cost accounting. The added parameters are two 64-unit gate MLPs and one linear attention head—roughly 30% more parameters than the trunk, and no additional optimization stages. The per-epoch store rewrite costs one extra teacher-forced pass. Measured wall-clock on two CPU threads, a CAMERA epoch takes 74.2s versus 26.6s for the plain trunk (2.79×); the comparison throughout is at *equal epochs* of training, and the modified model would be still more efficient under an equal wall-clock budget.

5 Experiments

Protocol. Both systems train for 24 epochs on the same corpus with the same seed and identical batch order; the trunk snapshot for checkpoint comparisons is epoch 4. We evaluate on validation (every epoch),

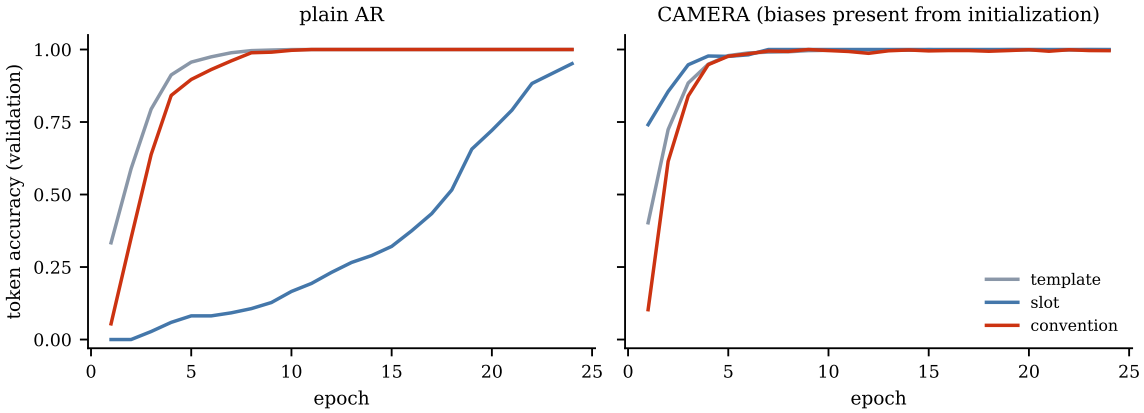


Figure 1: Training curves, validation accuracy per token class. Left: the plain trunk—lookup content (C), the never-stated house constants, is memorized within a few epochs, while copying stated values (S) plateaus near zero before a late transition. Right: the same trunk with the three biases present from initialization—copying is fast from the start, and the conventions are not harmed. Ground truth comes from the generator; no model ever sees these labels.

Table 2: Instruction-conditioning ablation (converged models, epoch 24). Hiding the instruction from attention separates what each model *memorized from code context* (survives) from what it *keyed on the instruction* (collapses). CAMERA's conventions degrade less than the baseline's because part of their load is carried by the episodic store; its slots collapse completely because they are carried by the copy path, which by construction has nothing left to read.

	template	slot (seen)	convention
AR	100%	93.5%	100%
AR, instruction hidden	52.1%	5.3%	39.9%
CAMERA	99.9%	100%	99.6%
CAMERA, hidden	63.9%	2.7%	48.6%

test-seen, test-novel, and per rarity tier, and report per-class token accuracy, contamination rates, gate statistics, store composition, and an instruction-conditioning ablation (every instruction token replaced by `PAD` and hidden from attention).

5.1 What plain next-token training learns slowly

The left panel of Figure 1 shows the plain trunk's validation accuracy by token class. Lookup content is fast: convention accuracy reaches 84.2% by epoch 4, because once family membership is identified, the constant is a table entry. Copying is slow: slot accuracy is 5.9% at epoch 4 and 28.9% at epoch 14, climbing to 95.1% only by epoch 24. The slow capability is precisely the one that requires an *operation* rather than a lookup: under this capacity and data regime, gradient descent builds the copying circuit late—after, not before, it has memorized the lookup content. Early in training the trunk also misattributes memorized constants across families (leak 8.9% at epoch 4), the signature of lookups retrieved under blurred keys.

The converged trunk eventually reproduces every training slot value (93.5%) and every convention (100%). Is that a copying operation? No. Novel slot values—stated in test instructions, absent from all training targets—are produced at only 4%, at every epoch of training. And when the instruction is hidden (Table 2), slot accuracy collapses to 5.3%, convention accuracy falls to 39.9%, and the contamination rate jumps from 0% to 31.4%: stripped of the key, the model's convention lookups blur across families. The trunk memorized instruction-keyed input–output mappings. The instruction serves as a retrieval key that selects among memorized constants and values; it is not read as a source to copy from.

Table 3: Main results. Per-class next-token accuracy on test-seen and test-novel splits at the epoch-4 checkpoint and at convergence (epoch 24). Both systems trained from scratch under identical conditions. Leak is the cross-family contamination rate.

	test-seen			novel	
system	T	S	C	S (novel values)	leak
AR @4	0.914	0.049	0.839	0.000	0.092
CAMERA @4	0.950	0.961	0.950	0.910	0.018
AR @24	1.000	0.935	1.000	0.040	0.000
CAMERA @24	0.999	1.000	0.996	1.000	0.002

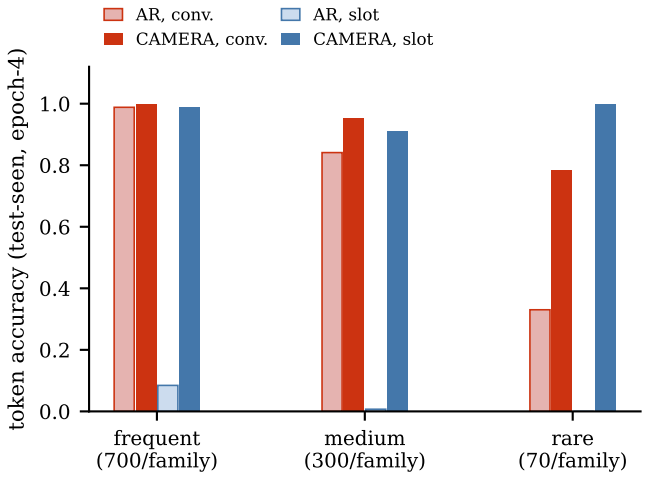


Figure 2: Epoch-4 checkpoint: convention (red shades) and slot (blue shades) accuracy by data-rarity tier for the plain trunk and CAMERA. The gap is largest exactly where training data is scarce.

5.2 The biases make next-token prediction fast

The right panel of Figure 1 shows the same trunk retrained with the biases present from initialization. The slot-copying plateau disappears: copying stated values is high from the first epochs (97.8% by epoch 4 against 5.9% for the plain trunk at the same epoch), and the model crosses the plain trunk's *final* copying accuracy (95.1%) within 4 epochs. Conventions are not sacrificed for this speed—they are learned at least as fast as in the baseline (94.8% vs. 84.2% at epoch 4)—and template grammar is unchanged. Nothing is traded away; the biases add pathways that make the hard capability easy.

The checkpoint rows of Table 3 quantify this at the epoch-4 snapshot: CAMERA reaches 96.1% on seen-value copying and 91% on novel values, where the plain trunk at the same epoch sits at 4.9% and 0%—and where it still sits at 4% on novel values after 24 epochs, since no amount of rehearsal produces a value the model was never trained to emit. At convergence both systems agree on the lookup content (99.6% vs. 100% conventions), and CAMERA retains its copying advantage (100% vs. 93.5% seen; 100% vs. 4% novel).

Data scarcity is where the bias pays most early (Figure 2). At epoch 4 the plain trunk recalls conventions of frequent families at 98.8% but rare families at only 33%—a clean scarcity gradient—while CAMERA reaches 78.3% on the rare tier, and 100% on rare-tier slot copying against 0% for the plain trunk. The store's entries are roughly tier-independent per training example (4.49 for frequent vs. 4.21 for rare families), yet its composition over-represents exactly the failure-prone classes (Section 5.4): a rare family with 10× less data ends up over-represented in memory because its facts are what keeps failing.

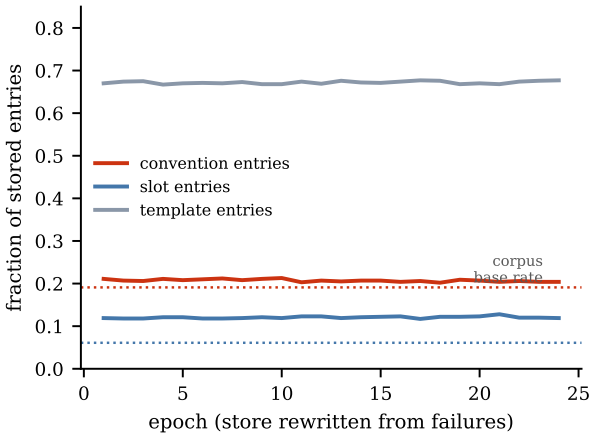


Figure 3: What the episodic store keeps over training (loss-based writes, rewritten each epoch). The store over-represents the failure-prone classes —slot entries sit at roughly twice their corpus base rate and conventions modestly above (dotted lines)—and under-represents the easy template positions; the composition is stable because the criterion always fills the capacity with whatever the current model still fails.

5.3 Generalization: copying what was never rehearsed

Novel slot values are the sharpest test of whether a model acquired a copying operation, because they cannot be produced by any amount of memorization: they appear in test instructions but in no training target. The plain trunk never exceeds 4% on them, at any epoch. The copy path is what closes this wall in the modified model: CAMERA produces novel values at 91% by epoch 4 and 100% at convergence. The pointer's attention itself is accurate—its top-1 attention target equals the ground-truth token on 98.8% of novel-slot positions at epoch 4.

Under the instruction ablation (Table 2), the modified model's slot accuracy collapses from 100% to 2.7%, with novel values unreachable by construction (no instruction positions remain for the pointer to read; γ is forced to 0). This confirms the copy pathway reads the instruction rather than the store: the store's entries, like the trunk's weights, hold only training-time values.

5.4 The gates and the store route themselves

No module ever sees a token role, so any division of labor in the trained system is emergent. Three observations. First, the store's composition is failure-driven, exactly as the write criterion implies (Figure 3): slot positions—the class the model fails on longest—occupy 12.1% of the entries at epoch 4, roughly twice their corpus base rate (6.1%), while easy template positions are under-represented (0.667% vs. 74.8%); the composition is stable across training because the criterion always fills the capacity with whatever the *current* model still fails. Retrieval quality matches: the store's top-1 hit rate on convention positions is 60.4% at the snapshot. Second, the copy gate routes stated-value positions to the pointer from the first epochs and tightens its hold as training proceeds: at epoch 4, γ is 90.1 on slots versus 0.8 on conventions, and by epoch 24 the copy gate owns slots outright ($\gamma = 1.00$, with 1.00 on the rare tier). Third, the two gates hand positions over to each other as the trunk learns: the store gate's weight on slots falls from 0.77 at epoch 4 to 0.28 at epoch 24—exactly as the copy gate takes them over—while it retains conventions (0.55 to 0.56), where episodic values remain useful longer. The division of labor that we never specified—copying for stated values, episodic recall for still-unlearned facts, trunk weights for grammar and internalized style—emerges from the loss alone.

5.5 Honest costs and blind spots

Three qualifications. First, the biases cost compute: a CAMERA epoch takes 74.2s versus 26.6s for the plain trunk (2.79×), dominated by the extra store-refresh pass and retrieval. The speedup in learning is far

larger than this overhead—the modified model exceeds the baseline's *final* copying accuracy at epoch 4—but at matched wall-clock the gap would be narrower. Second, the young store is briefly noisy: at epoch 1 the modified model's contamination rate (6.1%) is slightly above the baseline's, because the first store's keys come from an almost untrained trunk; by epoch 4 the ordering reverses (1.8% vs. 8.9%) and stays reversed. Third, at convergence the plain trunk is marginally better on the one axis its rehearsal can exhaustively search—rare-family conventions (100% vs. 97.4%)—while the modified model is at or near perfect everywhere else, including novel-value copying (100%), which remains beyond the baseline entirely. A final blind spot is inherited from loss-only training: the gates can only observe failures that occur on training positions. Values that appear only at test time are covered because the copy path reads the context rather than the weights, but any capability whose failures never surface during training would not be routed anywhere.

6 Discussion

Slow learning is a bias problem. The plain trunk's failure to copy is architectural, not informational: its hidden states demonstrably contain the instruction content (masking the instruction collapses slot accuracy), and a single trained pointer reading the same states achieves 98.8% top-1 hits on novel values from epoch 4. Gradient descent, under this capacity regime, finds lookup solutions first and builds the copying circuit late. This is a small-scale instance of a general pattern: context-mediated circuits such as induction heads form late and unreliably (Olsson et al., 2022), and Chan et al. (2022); Raventós et al. (2023) show that distributional properties gate their emergence. An explicit pointer converts "hope that the circuit forms" into an architectural commitment—and in these runs it converts twenty epochs of plateau into immediate progress.

Why the biases do not compete. They hold different things. The store maps contexts to training-time values; the pointer maps positions to context tokens. Their failure modes are complementary—the store cannot emit a value that was never written, and the pointer cannot emit anything the instruction does not state—and the gates negotiate the boundary from loss alone. No role labels were used anywhere in training, so the same machinery applies to corpora whose token roles are unknowable.

Limitations and scope. The suite is synthetic and the trunk is tiny ($\sim$100k parameters), chosen so that token-level ground truth exists and every mechanism is inspectable; nothing here establishes scale behavior. Each configuration is a single run (one seed), so we report central tendencies, not distributions. The copy head receives the instruction span as a structural input (the sequence's `SEP` boundary), which real systems would need to identify without such markers. The store is rewritten from training-set failures, so its writes reflect teacher-forced losses rather than online-decoding failures. Greedy decoding abstracts away sampling effects. And the write criterion's per-family median uses family identities that a real system would have to infer; a global median is the label-free fallback. We view the contribution as a controlled decomposition with clean attribution—a template for asking mechanism-level questions about what to add to a plain next-token predictor, and for measuring which added bias pays for which failure.

7 Conclusion

On an instruction-to-code suite with token-level ground truth, plain next-token training acquires lookup content quickly but copying slowly, learns scarce facts last, and never builds an operation for values that appear only at test time. Retraining the same trunk from the same budget with three biases present from initialization—an episodic store rewritten each epoch from the model's own failures, per-position gates trained from the loss alone, and a pointer copy path over the input—removes the copying plateau, recovers scarce families early, and makes novel test-time values producible (91% by epoch 4 against a lifetime maximum of 4% for the baseline). The gates discover the division of labor from the loss alone. What to store, what to copy, and when to trust each can be learned; making them explicit mechanisms is what makes a plain next-token predictor fast where it is slow, and capable where it is structurally not.

Reproducibility statement

All experiments run on CPU in ∼50 minutes for both systems. The archive contains the corpus generator with token-role ground truth, the full source for both systems (trunk, store, gates, copy head), the runner, the run log, all reported numbers in one `metrics.json` file, figure generation code, and this paper's sources. Every number in the text is taken from that file.

References

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws. *arXiv preprint arXiv:2404.05405*, 2024.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eric Rutherford, Katie Millican, George van den Driessche, Jack B. Rae, et al. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning (ICML)*, 2022.

Stephanie C. Y. Chan, Adam Santoro, Andrew J. Lawson, Ishita Dasgupta, Andrew Lampinen, Samuel J. Gershman, and Felix Hill. Data distributional properties drive emergence of in-context learning in transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Vitaly Feldman. Does learning require memorization? A short tale about long tails. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2020.

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, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, 2019.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations (ICLR)*, 2020.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Memory-based model editing at scale. In *International Conference on Machine Learning (ICML)*, 2022a.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. Fast model editing at scale. In *International Conference on Learning Representations (ICLR)*, 2022b.

Neel Nanda, Andrew Lee, and Martin Wattenberg. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*, 2023.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova Novinsky, Catherine Rager, Tom Conmy, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Allan Raventós, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-Bayesian in-context learning for regression. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. In *International Conference on Machine Learning (ICML)*, 2016.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Kushal Tirumala, Armen Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. Memorization without overfitting: Analyzing the training dynamics of large language models. *arXiv preprint arXiv:2205.10770*, 2022.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations (ICLR)*, 2017.

A Corpus Samples with Token Roles

```
INSTRUCTION: Implement scale_num:  multiply the input by 3 and round the result.
def scale_num(x):  [S]
  """scale_num:  multiply the input by 3 and round."""  [S,T]
  x = float(x)  [C]
  return round(3 * x, 3)  [S,T,--,C]
Novel-value test:  Implement scale_batch:  multiply the input by 20 and round the result.  --
the 20 is in no training target; copying it requires reading the instruction.
```

Figure 4: Token roles in a corpus sample. Template (T) tokens are family grammar; slot (S) tokens restate instruction values; convention (C) tokens are the family's unstated house style (`float()` cast, rounding precision 3).

B Hyperparameters and Protocol

C Store and Gate Statistics

D Qualitative Generations

Greedy decodings for one example of every family are included in the archive (`generated_samples.txt` in the results directory), with the mean gate weights recorded per generation. The decisive qualitative evidence for the copy path is quantitative, on the novel-value split: the pointer reads the instruction directly and

Table 4: Hyperparameters. Both systems: seed 11, 24 epochs, identical batch order; trunk snapshot for checkpoint tables at epoch 4.

parameter	value
d_{model} / layers / heads / d_{ff}	96 / 2 / 3 / 192
dropout	0.1
optimizer	AdamW
learning rate	3×10^{-4}
batch size	64
store capacity / retrieval K / temperature β	20,000 / 8 / 10
store rewrite	every epoch, from persistent failures
gate hidden width / α budget	64 / 0.02
copy head	1 linear-attention head over instruction positions

Table 5: Top: store composition by token role (the write criterion never sees the roles). Bottom: mean gate weights on test-seen positions. The copy gate owns slot positions from the first epochs; the store gate hands slots over to it as the trunk learns, while retaining conventions.

role	store	corpus base rate
convention (C)	21.1% / 20.4%	19.1%
slot (S)	12.1% / 11.9%	6.1%
template (T)	66.7% / 67.7%	74.8%
	epoch 4	epoch 24
copy gate γ: slot	0.90	1.00
copy gate γ: convention	0.01	0.00
store gate α: slot	0.77	0.28
store gate α: template	0.44	0.44
store gate α: conv. (rare tier)	0.61	0.63

transfers the stated value into signature, docstring, and body slot positions at 91% token accuracy by epoch 4, while the plain trunk—whose predictions there must come from memorized training values, since no store entry or weight can hold an unstated value—achieves at most 4% at any epoch.