PERSISTENT-CACHE COMPLEXITY: A FINITE-STATE AND COMMUNICATION-THEORETIC VIEW OF CAUSAL DECODERS

Anonymous authors
Paper under double-blind review

ABSTRACT

How much memory does a causal decoder need to retain enough information to answer queries about an input it has already processed? We study this question by counting all input-dependent information retained for future computation. For independent, uniformly distributed records compressed before an independent query is revealed, we derive memory lower bounds for a target average recall error. Under uniform queries, coding constructions asymptotically match these bounds, characterizing the optimal tradeoff between retained memory and recall accuracy. When queries have unequal frequencies, our bounds explain how memory can favor frequently requested records. We show that encoding a summary can outperform storing a fixed subset of records exactly, although this advantage at a fixed storage fraction decreases as the number of possible record values grows. Our analysis counts selection indices and reusable randomness, and distinguishes memory retained after compression from peak memory and physical cache storage. Experiments with finite-state models and Transformer caches on formal language recognition, synthetic associative recall and string retrieval illustrate these discovered tradeoffs and the importance of query timing.

1 INTRODUCTION

Decoder-only Transformers reuse earlier keys and values to process tokens in a causal manner (Vaswani et al., 2017). However, as contexts grow, this persistent key-value (KV) state becomes a major serving cost (Jiang et al., 2026). Recent methods have tried to solve this problem by retaining attention sinks and recent tokens (Xiao et al., 2024), prioritizing heavy hitters (Zhang et al., 2023), selecting positions from an observation window (Li et al., 2024), or correcting sparse attention (Willette et al., 2025). The effectiveness of these methods depends on which information is needed by subsequent requests.

However, counting retained positions does not measure this information. Deleting a record's own KV entries does not mean it is erased, as its value may have been encoded elsewhere. Meanwhile, a retained entry may contribute little to future answers. Thus, the relevant object is the *complete persistent state, through which a prefix can affect its continuation*. We ask: *how many persistent bits must a causal decoder preserve to distinguish the future continuations relevant to a task?*

We study this question through *persistent-cache complexity*. Working in the classical streaming model with length-dependent transition rules, we define this measure as the number of persistent state bits that survive between processing steps. Our model permits unrestricted computation, isolating what must be remembered from how efficiently it can be computed. A cache snapshot acts as a one-way message from the computation on an input prefix to the computation on its continuation (Kushilevitz & Nisan, 1997). This connection transfers communication lower bounds to cache requirements. For exact deterministic computation, it also gives a finite-length version of future equivalence (Nerode, 1958): two prefixes must remain distinguishable whenever the same continuation would require different outputs.

Our main results concern independent uniform q-ary records compressed before a query is known, with query indices independent of the database. Using Fano's inequality and random-access bounds (Nayak, 1999, Cover & Thomas, 2006), we lower-bound the memory needed for a target average

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

recall error. The bound permits unequal record errors, explaining how memory can favor frequently queried records. Rate-distortion constructions match it asymptotically under uniform queries and when query probabilities are constant within each of a fixed number of growing groups. Coded summaries can outperform fixed literal retention by preserving partial information about many records. Using perfect codes, we construct a 64-bit summary of 128 binary records with 86.12% expected accuracy under uniform queries, versus 75% for retaining half and guessing the rest. At a fixed storage fraction, the large-database advantage of optimal summaries over literal retention scales as $1/\log_2 q$ as the alphabet grows.

Applying these results requires complete accounting and attention to query timing. Data-dependent selection indices can carry information beyond retained values. A cache prepared before a query must support unseen requests, whereas compression afterward can specialize to that request. Any source cache retained for reselection must also be counted. Therefore, we distinguish the retained summary, peak persistent state, and complete physical storage. Our recall frontier characterizes the first and lower-bounds the others under matching assumptions. Reusable private seeds also count, unlike freely accessible public randomness. Our cache-state packing proof for equality recovers the classical separation between these conventions (Kushilevitz & Nisan, 1997, Newman, 1991). These results complement attention-specific lower bounds (Haris & Onak, 2025) by measuring task-level information requirements, rather than the resources to reproduce a particular attention computation.

Four diagnostic experiments link these distinctions to concrete models. Formal language tasks examine exact recognition and approximation under limited persistent memory. Associative recall compares literal retention with learned summaries. Transformer KV cache experiments then examine the effect of revealing queries before or after compression. Finally, SCBench's String Retrieval tasks (Li et al., 2025) test retrieval under cache reduction, while transfers of record-level KV entries probe whether retained information causally affects the answer. These experiments distinguish task-level memory requirements from how effectively representations preserve and use that information.

2 Theoretical Framework

Input and tasks. We model the input as a sequence over a nonempty finite alphabet Σ, corresponding to a finite token vocabulary. For every input length n, let Σ^n denote the set of all length-n strings over Σ. A task family $\mathcal{T} = (T_n)_{n\geq 0}$ consists of target functions $T_n : \Sigma^n \to \mathcal{Y}_\mathcal{T}$, where $\mathcal{Y}_\mathcal{T}$ is a finite output set. Each T_n is defined on every length-n input and specifies the required answer.

We focus on language recognition and record retrieval within this model, because they make the information required for future answers exactly identifiable. For language recognition, a language $L \subseteq \Sigma^\star$ defines the target $T_n(x) = \mathbf{1}\{x \in L\}$ with output set $\{0, 1\}$. For record retrieval, let $m \geq 1$ and $q \geq 2$ be integers. A database $X = (X_1, \ldots, X_m) \in [q]^m$ contains m records, each taking one of q possible values, where $[q] = \{1, \ldots, q\}$. A query specifies an index $i \in [m] = \{1, \ldots, m\}$, and the required answer is X_i. The database and query are encoded as input blocks over Σ, with their order specified by the task. Thus, m counts records, whereas n counts input symbols. For a random query index $I \in [m]$, let π denote its distribution, with $\pi_i = \Pr[I = i]$.

Persistent state. For every n and time $t \in \{0, \ldots, n\}$, the decoder has a finite set $\mathcal{C}_{n,t}$ of cache states, with a distinguished public initial state $C_{n,0} \in \mathcal{C}_{n,0}$. As n and t are known control parameters, information that can be derived from them is free. Otherwise, any input dependent data (e.g., phase, position, selection pattern) must be encoded in the cache. A deterministic state transition has the form $U_{n,t} : \mathcal{C}_{n,t-1} \times \Sigma \to \mathcal{C}_{n,t}$, while a randomized transition is a stochastic kernel $U_{n,t} : \mathcal{C}_{n,t-1} \times \Sigma \rightsquigarrow \mathcal{C}_{n,t}$. After the final symbol, an output map $A_n : \mathcal{C}_{n,n} \to \mathcal{Y}_\mathcal{T}$ produces the decoder's answer.

Conventions. Fresh randomness can be used freely by one transition and never accessed again. However, if a realized random value is needed by a later transition, that value must be encoded in the persistent state and is counted. We call this the *default private-state convention*. Under the private-state convention, the conditional law of the next state depends only on the current state, the current symbol, and the public parameters n, t. We separately refer to a *public-coin convention* when all transitions can access a common random tape for free. These models are not equivalent, as public randomness can reduce the amount of persistent state required by some tasks.

Transition tables, model parameters, and output maps may depend on n and t, but not on the input instance, and their description length is not charged. Consequently, the model is nonuniform and

computation-unbounded: it does not measure the time required to compute a transition or whether that transition is realizable by a fixed Transformer. Its purpose is to measure how much input-dependent information must remain available after each token boundary.

Correctness and cache complexity. For a decoder D, let the cache size at t be $S_D(n,t) = \lceil \log_2 |\mathcal{C}_{n,t}| \rceil$. Thus, cache complexity is determined by the number of persistent states, rather than by a particular physical representation. With $C^D_{n,n}(x)$ being the final state reached by decoder D on input x, its output is $\hat{T}^D_n(x) = A_n(C^D_{n,n}(x))$. An exact deterministic decoder satisfies $\hat{T}^D_n(x) = T_n(x),\ \forall x \in \Sigma^n$, while a randomized one has worst-case error at most ε if $\Pr[\hat{T}^D_n(x) \neq T_n(x)] \leq \varepsilon,\ \forall x \in \Sigma^n$. Unless an input distribution is specifically defined, probability is only over decoder randomness. For a task family $\mathcal{T}$, define the cache complexity as

$$K^{\mathrm{det}}_{\mathcal{T}}(n) = \min_{D \text{ computing } T_n \text{ exactly}} \left(\max_{0 \leq t \leq n} S_D(n,t) \right). \tag{1}$$

Under the private-state randomized convention, define $K^{\mathrm{priv},\varepsilon}_{\mathcal{T}}(n)$ analogously using private-state randomized decoders with worst-case error at most ε on every input in Σ^n. Define $K^{\mathrm{pub},\varepsilon}_{\mathcal{T}}(n)$ by the same minimization when transitions may consult a common public random tape, independent of the input, without storing it. State cardinalities are bounded uniformly over tape realizations. For language recognition, we write K_L as shorthand for $K_{\mathcal{T}}$, where L is the target language.

Note that the model imposes no restrictions on the computation used to update or decode the cache. At each step, a decoder may choose any encoding of the information available to it, subject to the stated constraints. Consequently, the theoretical results below apply to every algorithm satisfying their assumptions, regardless of its architecture, computational power, or compression strategy.

2.1 Cache-to-Communication Transfer and Query Timing

A cache snapshot is the complete message passed from a prefix to its continuation. This is the classical streaming-to-communication interface (Alon et al., 1996, Kushilevitz & Nisan, 1997).

Theorem 2.1 (Transfer and exact query timing)**.** ***(a) Transfer.*** *Fix an input length n, a split $t \in \{0, \ldots, n\}$, and an integer $s \geq 0$. Define $F_{\mathcal{T},n,t}(u,z) = T_n(uz)$ for $u \in \Sigma^t$ and $z \in \Sigma^{n-t}$. If an ε-error causal decoder has at most 2^s states after t symbols, then $F_{\mathcal{T},n,t}$ has an ε-error one-way protocol with an s-bit message under the same private-state or public-coin convention. The proof is given in Appendix B.1.*

(b) Query timing. *For exact recovery of X_i on every (X,i) in the record retrieval task, the minimum state at the indicated boundary is $B^\star = \lceil m \log_2 q \rceil$ with processed prefix X and remaining input i. If the prefix is i and X remains, $B^\star = \lceil \log_2 m \rceil$. After both blocks, exactly $\lceil \log_2 q \rceil$ bits suffice and are necessary. These are boundary budgets, and other stages may use more memory. The proof is given in Appendix B.2.*

In Theorem 2.1, part (a) transfers one-way communication lower bounds to the information that a causal decoder must retain at a given boundary. Consequently, established communication results impose cache requirements on every decoder satisfying the same assumptions. Part (b) illustrates this connection through query timing: changing when the query becomes available changes the communication problem across the boundary and, together with matching constructions, determines the exact cache requirement. For extensions, Corollary B.2 determines minimum peak memory for exact deterministic decoding (Nerode, 1958, Aaronson, 2004). Corollary B.12 extends timing to repeated independent requests from the same law, preserving mean error per request without extra memory and showing how known queries can reduce storage.

2.2 Recall under a Query Law: Converse and Frontier

For record retrieval, let $S^\star_\pi(m, \bar{e})$ be the minimum cache size in bits after encoding a uniform database $X \in [q]^m$ and before the query arrives. The cache must support answers with population error at most $\bar{e}$ under query law π and the private-state convention. The encoder may inspect all of X before forming the cache. Thus, $S^\star_\pi$ measures the retained summary, whereas $K_{\mathcal{T}}$ minimizes the largest persistent state used throughout a causal computation (see formalities in Definition B.5).

Three levels of memory. *(i) Boundary information* $S^\star_\pi(m,\bar{e})$ minimizes the state surviving compression over unrestricted encoders of the full database. Theorems 2.2 and B.6 characterize this quantity asymptotically under their respective query-law assumptions. Proposition 2.4 supplies explicit attainable points, without asserting their optimality at finite m.

(ii) Peak persistent state of a causal decoder D is $P_D(n) = \max_{0\le u\le n} S_D(n,u)$, and $K_\mathcal{T}$ minimizes this peak under its stated correctness and randomness conventions. Part (a) of Theorem 2.1 and Corollary B.2 and Proposition 2.5 address these causal state requirements. Because the peak includes state retained during encoding, a small final summary does not guarantee a small peak. To limit encoding memory, fixed-size perfect-code blocks can be processed one at a time, retaining only the current block and the code indices of completed blocks.

(iii) Physical cache storage connects the abstract state measures to the bits/bytes used by an implementation. It includes the complete stored state, such as positions, selection indices, quantization scales, and reusable seeds. Proposition 2.3 accounts for retained values and their selection indices, while Remark B.14 translates information bounds into constraints on KV storage and persistent metadata. Let $M_D(n,u)$ be the worst-case byte capacity of a fixed-length serialization of that state at time u. For a private-state implementation meeting the stated recall target at the pre-query boundary t, the definitions give

$$S^\star_\pi(m,\bar{e}) \le S_D(n,t) \le P_D(n) \le 8 \max_{0\le u\le n} M_D(n,u), \quad S_D(n,t) \le 8M_D(n,t). \tag{2}$$

The bound $S^\star_\pi(m,\bar{e}) \le S_D(n,t)$ follows because the boundary state is a valid cache, and $S_D(n,t) \le P_D(n)$ follows by definition. Counting at most $2^{8M_D(n,u)}$ serialized states at each boundary gives $S_D(n,t) \le 8M_D(n,t)$ and, after maximizing over u, $P_D(n) \le 8\max_{0\le u\le n} M_D(n,u)$. Minimizing the peak preserves the lower bound for $K_\mathcal{T}$ under the same recall task and private-state convention, provided its correctness guarantee implies the stated population-error target. However, storage at t alone does not upper-bound the peak.

Theorem 2.2 (Recall converse and asymptotic frontier)**.** *Write $c_q = 1 - 1/q$ and $f_q(a) = \log_2 q - h_2(a) - a\log_2(q-1)$, where $h_2(a) = -a\log_2 a - (1-a)\log_2(1-a)$ is the binary entropy function, with $0\log_2 0 = 0$. Let X be uniform on $[q]^m$. An encoder ψ forms the cache $C = \psi(X,U)$ using randomness U independent of X. The cache contains all persistent randomness and has at most 2^S possible values. For each $i \in [m]$, a decoder ϕ_i estimates X_i as $\widehat{X}_i = \phi_i(C, Z_i) \in [q]$, using fresh randomness Z_i independent of (X,C). Its population error is $e_i = \Pr[\widehat{X}_i \ne X_i]$.*

(a) Coordinatewise bound and maximum a posteriori (MAP) postprocessing. *Let $\mathrm{supp}(C)$ denote the set of possible cache values. For each i, define the minimum error over deterministic decoders ϕ by $e_i^\star = \min_{\phi:\mathrm{supp}(C)\to[q]} \Pr[\phi(C) \ne X_i]$. Then, $e_i^\star \le \min\{e_i, c_q\}$ and $S \ge S_{\mathrm{Fano}}(e^\star) := \sum_{i=1}^m f_q(e_i^\star)$. Also $S \ge \sum_i f_q(e_i)$ for the exact decoder errors. If $e_i^\star \le u_i \le c_q$ for every i, then $S \ge \sum_i f_q(u_i)$. For an integer bit budget, these inequalities allow a ceiling around the whole sum, with no separate rounding of coordinate contributions.*

(b) Query-weighted bound and optimizer. *Let the query index I be sampled from π, independently of (X,C) and the fresh decoding randomness. If $\sum_i \pi_i e_i \le \bar{e}$ for a target $0 \le \bar{e} \le c_q$, then*

$$S \ge \mathcal{R}_\pi(\bar{e}) := \min_{\substack{0\le a_i\le c_q \\ \sum_i \pi_i a_i \le \bar{e}}} \sum_{i=1}^m f_q(a_i). \tag{3}$$

The alphabet size is implicit in $\mathcal{R}_\pi$. For uniform queries, $\mathcal{R}_{\mathrm{unif}}(\bar{e}) = m f_q(\bar{e})$. If every $\pi_i > 0$ and $0 < \bar{e} < c_q$, the unique optimizer is

$$a_i^\star = (q-1)/(q-1+2^{\lambda\pi_i}), \qquad \sum_{i=1}^m \pi_i a_i^\star = \bar{e}, \tag{4}$$

where the second equation determines a unique $\lambda > 0$. For $q = 2$, this gives $a_i^\star = (1+2^{\lambda\pi_i})^{-1}$ and $f_2(a) = 1 - h_2(a)$. Coordinates with $\pi_i = 0$ may be assigned $a_i^\star = c_q$. We extend $\mathcal{R}_\pi(\bar{e}) = 0$ for $\bar{e} \ge c_q$, since the constant chance-level error vector is then feasible.

(c) Uniform-query achievability. *Fix q and $\bar{e} \in (0, c_q)$. Write $S^\star_{\mathrm{unif}}(m,\bar{e})$ for $S^\star_\pi(m,\bar{e})$ with $\pi_i = 1/m$. For every $\gamma > 0$ and sufficiently large m, $m f_q(\bar{e}) \le S^\star_{\mathrm{unif}}(m,\bar{e}) \le m f_q(\bar{e}) + \gamma m$. A deterministic cache index and deterministic coordinate decoders attain the upper bound. The proofs are in Appendices B.6 and B.7.*

Theorem 2.2 helps determine the memory needed for a target recall error and guides how accuracy is allocated across records. Under its assumptions, parts (a) and (b) give memory lower bounds that every encoder and decoder must satisfy, so additional computation cannot compensate for insufficient stored information. In particular, exact recall of every record requires at least $m \log_2 q$ bits. For nonuniform queries, the lower-bound optimizer in part (b) assigns smaller errors to frequently requested records because those errors contribute more to average loss, providing a guide for allocating recall accuracy. For uniform queries, part (c) shows that deterministic summaries asymptotically attain the lower bound. Consequently, at fixed alphabet size and fixed error below chance, the required cache memory grows linearly with the number of records. The attainable bound provides a benchmark for evaluating compression methods, while coded summaries can improve on fixed literal retention by preserving partial information across records (Corollary B.9).

2.3 Selection Accounting and Cache Reuse

Selecting literal records can preserve information through both their values and their indices. The following statement isolates that policy class.

Proposition 2.3 (Accounting for selected records)**.** *Let $0 \le k \le m$ be an integer, X be uniform on $[q]^m$, and $R \subseteq [m]$ have size k, possibly chosen from X and independent random coins. Suppose the complete cache is exactly $C = (R, X_R)$, with values listed in increasing index order and no additional surviving state. For an independent uniform query, define $B_{\mathrm{sel}} = \left\lceil k \log_2 q + \log_2 \binom{m}{k} \right\rceil$. Then, $|\operatorname{supp} C| \le \binom{m}{k} q^k$, so C admits a representation of at most B_{sel} bits. Every decoder satisfies $\mathrm{Acc} \le 1 - f_q^{-1}(\min\{B_{\mathrm{sel}}/m, \log_2 q\})$, where f_q^{-1} is taken on $[0, c_q]$. If R is independent of the entire database X, the optimal accuracy is exactly $A_{\mathrm{lit}} = 1/q + c_q k/m$. More generally, let $\epsilon_{\mathrm{sel}} = \mathrm{TV}(P_{X,R}, P_X P_R)$ denote the total variation distance between the joint distribution of (X, R) and the product of its marginals. The optimal decoding accuracy $\mathrm{Acc}_{\mathrm{MAP}}$ then satisfies $0 \le \mathrm{Acc}_{\mathrm{MAP}} - A_{\mathrm{lit}} \le \epsilon_{\mathrm{sel}}$. The proof appears in Appendix B.10.*

Proposition 2.3 shows that, for a uniform database and an independent uniform query, value-independent selection of k literal records cannot outperform the literal-retention baseline A_{lit}, regardless of the selection algorithm or decoder. The guarantee applies when the complete cache contains only the selected indices and values, $C = (R, X_R)$. More generally, the accuracy gain from value-dependent selection is at most ϵ_{sel}, so weak dependence between the selected set and record values permits only a small improvement over the literal baseline. For extensions and related results, Example B.11 shows why value-dependent indices must be counted, since they can encode omitted values. For nonuniform queries, the optimal fixed k-record literal cache retains the most frequently requested records (Corollary B.10). Independent queries from the same law can reuse one cache at unchanged size and mean error per request (Corollary B.12).

2.4 Explicit Finite Summaries from Perfect Codes

We further consider binary record retrieval from a uniform database with independent uniform queries. The following proposition uses perfect codes to construct caches before queries arrive and gives explicit formulas for cache size and population error.

Proposition 2.4 (Perfect-code summaries)**.** *Let X be uniform on $\{0,1\}^m$, and let I be an independent uniform query in $[m]$. For a summary of S bits, write $\rho = S/m$ for the memory fraction and $\bar{e} = \Pr[\widehat{X}_I \ne X_I]$ for population retrieval error. A binary perfect code of length n, cardinality 2^k, and radius t encodes each block by its nearest-codeword index in k bits, with expected Hamming loss $\mu(n,t)$ given by* (28)*. With codebooks and disjoint block assignments fixed publicly and independently of X, compositions with literal and guessed bits attain* (29)*.*

When m is divisible by the respective code length, Hamming $[2^r - 1, 2^r - 1 - r]$ codes with integer $r \ge 2$ attain $(\rho, \bar{e}) = (1 - r/(2^r - 1), 2^{-r})$, and the binary Golay $[23, 12]$ code attains $(\rho, \bar{e}) = (12/23, 127/1024)$. When $2t + 1$ divides m, the repetition code $R_{2t+1} = \{0^{2t+1}, 1^{2t+1}\}$, with integer $t \ge 1$, attains $\rho = 1/(2t+1)$, $\bar{e} = 1/2 - \binom{2t}{t}/(2^{2t+1})$. Appendix B.9 gives the full argument and endpoint accounting.

Proposition 2.4 implies that deterministic summaries can outperform fixed literal retention at the same finite bit budget, for a uniform binary database and an independent uniform query. Because

each code index preserves partial information about several records, accurate retrieval does not require retaining every queried record literally. For Transformer KV compression, this motivates preserving information across records before removing cache positions. A removed position need not imply complete loss of its record if other retained states encode relevant information. The supporting analysis shows the 64-bit construction for 128 binary records in Appendix B.9 which attains 86.12% accuracy, compared with 75% for fixed literal retention. The same appendix also shows that Hamming summaries match the recall error lower bound to first order as the memory fraction approaches one. For a broader comparison, Corollary B.9 characterizes the asymptotic advantage of optimal summaries over fixed literal retention under uniform queries and explains its dependence on alphabet size.

2.5 Randomness Conventions: Equality Separation

Let the language $\mathrm{EQ}_m = \{x\#y : x, y \in \{0,1\}^m,\ x = y\}$. The private-state and public-coin conventions already differ on this classical recognition task.

Proposition 2.5 (Private-state and public-coin equality)**.** *For every fixed $\delta \in (0, 1/2)$, as $m \to \infty$,*

$$K^{\mathrm{priv},\delta}_{\mathrm{EQ}_m}(2m+1) = \Theta_\delta(\log m), \qquad K^{\mathrm{pub},\delta}_{\mathrm{EQ}_m}(2m+1) \le \lceil \log_2(1/\delta) \rceil + O(1). \tag{5}$$

More precisely, a private-state cache of S bits immediately after the first block satisfies $S \ge \log_2 m - \log_2 \left[\log_2(1 + 1/(1-2\delta))\right]$. The constants in the public-coin upper bound are absolute. Full proof appears in Appendix B.5.

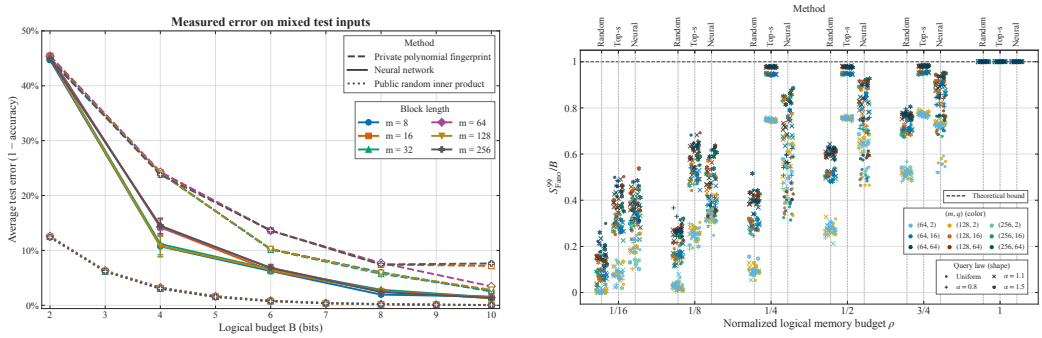
Proposition 2.5 shows that the randomness convention changes the memory required for equality. At fixed worst-case error $0 < \delta < 1/2$, private-state memory grows logarithmically with input length, while public randomness available without a storage charge permits constant memory (Kushilevitz & Nisan, 1997, Newman, 1991). Consequently, reusable private seeds must count toward the cache budget. For broader cache theory, Proposition B.3 collects memory bounds for regular languages, equality, and Dyck. Corollary B.13 and Remark B.14 give worst-case recall bounds and their KV-storage consequences, while Remark B.15 exhibits different memory growth rates through padded INDEX.

3 Experiments

3.1 Experiment 1: Persistent memory across formal languages.

Settings. We evaluate tasks with qualitatively different requirements of persistent memory. For regular languages, we use Tomita Grammar 7 (Tomita7) and binary divisibility by 31 (Mod31). An online neural controller receives one symbol and its previous B-bit state at each step, and only this state persists between symbols. The exact requirements are 3 bits for Tomita7 and 5 bits for Mod31. For equality $x\#y$, we compare deterministic and randomized finite-state mechanisms, including private polynomial fingerprinting, free public-coin random fingerprints, and learned discrete controllers. For Dyck-(2), we compare quantized GRUs with the one-pass randomized streaming construction of Magniez et al. (2014). We evaluate three seeds and several input lengths. The complete training and benchmark details are given in Appendices C.1.1, C.1.3 and C.1.5.

Results. *For Tomita7 and Mod31*, the learned controllers achieve exact recognition with 3 bits for Tomita7 and 5 bits for Mod31 at every tested length and seed. Exhaustive state traversal recovers exactly 5 and 31 reachable states, respectively, matching the future-equivalence characterization in Corollary B.2. *For equality*, Figure 1a compares error at matched budgets. Public-coin error approximately halves per added bit and varies little with block length. The observed public advantage supports the distinction in Proposition 2.5: free mask reuse supplies an additional resource, explaining why reusable private seeds must be counted. Neural gains over polynomial fingerprints illustrate how learning can preserve distinctions useful on the benchmark. Such gains are compatible with the exact-cache characterization, because low average error can coexist with collisions on distinguishing continuations. *For Dyck-(2)*, at every tested length, validation-based and test-based selection yield identical GRU budget ceilings for pooled error below 5% in every seed. Per-family ceilings change only by -32 bits at $n = 128$ and $+16$ bits at $n = 512$ when selecting on validation. Thus, the selected budgets remain close at every tested length, differing by at most 12.5% relative



(a) (b)

Figure 1: (a) Equality error versus persistent-memory budget. (b) Fano ratios S^{99}_{Fano}/B for Random and Top-s literal retention and grouped codebook on associative recall.

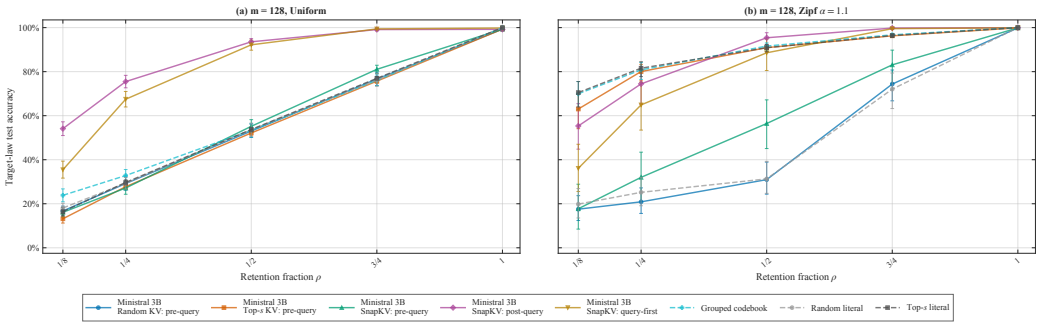


Figure 2: Associative-recall accuracy (64 databases, seed 47, $q = 16$). ρ is logical-memory ratio (grouped codebook), record-retention ratio (physical KV and literal), or prefix-token capacity ratio (SnapKV). Error bars show 95% paired database-bootstrap confidence intervals.

to test selection. Their overall growth is qualitatively consistent with the $\widetilde{\Theta}(\sqrt{n})$ memory regime in Proposition B.3. Detailed results are given in Appendices C.1.2, C.1.4 and C.1.6.

3.2 Experiment 2: Associative Recall Under a Fixed Memory Budget

Settings. We next test the associative recall results in Theorem 2.2 and Corollary B.10. Each instance contains $m \in \{64, 128, 256\}$ independent q-ary records, with $q \in \{2, 16, 64\}$, followed by a query revealed only after the database has been compressed. Queries are either uniform or Zipf distributed with $\alpha \in \{0.8, 1.1, 1.5\}$. We sweep normalized logical memory budgets $\rho \in \{1/16, 1/8, 1/4, 1/2, 3/4, 1\}$ and compare three representations: a fixed random subset of records, the top-s fixed subset prescribed by Corollary B.10, and a learned grouped-codebook summary whose packed indices contain exactly the same logical number of bits. For $q = 2$, we also evaluate perfect-code summaries from Proposition 2.4 at the same logical memory budgets. Each condition uses 256 databases, 16 stratified queries per database, and 3 seeds. The complete training and benchmark details are provided in Appendix C.2.1.

Results. Accuracy generally rises with ρ and falls with q, matching Corollary B.10 for literal retention. Under uniform queries, Random and Top-s perform similarly with weak m dependence, while grouped codebook gains most from jointly encoding binary records, qualitatively matching Corollary B.9's alphabet dependence. At $(m, q, B) = (128, 2, 64)$, three-seed mean accuracies are 86.00% (perfect codes), 75.28% (Random), 74.68% (Top-s), and 81.58% (grouped codebook). Perfect-code and literal results closely match the respective predictions of 86.12% and 75% (Proposition 2.4 and Corollary B.10), demonstrating finite-budget gains from preserving partial information across records. Yet grouped codebook outperforms perfect codes at $\rho \in \{1/16, 1/8\}$ for every tested m. Under Zipf queries, Top-s and grouped codebook improve with concentration and

m by favoring head ranks at the tail's expense; Random spreads accuracy across ranks. Following Theorem 2.2, 99% bootstrap upper error bounds yield conservative Fano estimates S^{99}_{Fano} and query-weighted $\mathcal{R}^{99}_{\pi}$, both within the logical budget B, consistent with the theorem; Figure 1b plots S^{99}_{Fano}/B. Across the grid, seed-averaged accuracies also remain at or below the Fano frontier, the maximum population accuracy permitted by the same information constraint at budget B under the target query law, supporting the memory–accuracy tradeoff for literal and learned representations. Remaining uniform-query gaps are compatible with Theorem 2.2(c), whose asymptotic guarantee does not assert optimality of tested finite summaries. Details appear in Appendix C.2.2.

3.3 Experiment 3: Query Timing and KV Cache Compression

Settings. We evaluate an actual Transformer KV cache and examine the query-timing distinction in part (b) of Theorem 2.1. Using the same associative recall benchmark, we test a frozen Ministral-3-3B-Instruct Liu et al. (2026) model under three processing orders. In **pre-query compression**, the database is compressed once and the resulting state $C(X)$ is reused across unseen queries. In **post-query compression**, the query is processed before compression, allowing the retained state to depend on both database and query (i.e., $C(X, i)$). Finally, in **query-first compression**, the query is available while the database itself is processed. We compare full KV retention with physical random and top-s record retention in the pre-query setting, and evaluate SnapKV (Li et al., 2024) under all three settings, using retained fractions $\rho \in \{1/8, 1/4, 1/2, 3/4\}$. The model is frozen and no task-specific training is performed. The full settings are given in Appendix C.3.1.

Results. Figure 2 compares LLM cache methods with Experiment 2's baselines under uniform and Zipf queries. Equal retention fractions do not imply equal physical memory budgets, since position buffers and continuation state also contribute to storage (Table 4). Counting these components follows Remark B.14. Physical Random and Top-s KV broadly track their literal controls, with Top-s benefiting from query concentration as Corollary B.10 predicts. Across the grid, accuracy generally improves with ρ, narrowing Top-s's gap from literal retention, and is highest for binary records on average. Larger m generally benefits Top-s under moderate Zipf skew and query-aware SnapKV. Across 144 tested settings, the median accuracy gains over pre-query SnapKV are 26.80 percentage points for post-query SnapKV and 20.88 for query-first SnapKV. The paired 95% confidence intervals for these gains lie entirely above zero in 139 and 133 settings, respectively. Under uniform queries, their median gains over literal Top-s are 30.91 and 22.56 percentage points across 36 settings. These timing gains agree qualitatively with part (b) of Theorem 2.1. The value-independent literal case of Proposition 2.3 also gives a conditional explanation for the proximity of pre-query SnapKV and Random. Applying that equality to SnapKV would additionally require value-independent selection and literal-only retained information. We provide full details in Appendix C.3.2.

3.4 Experiment 4: Retrieval and Causal Use of Long-Context Caches

Settings. The preceding experiments use synthetic data generated for this study. We now extend the evaluation to long shared contexts using the String Retrieval tasks of SCBench (Li et al., 2025), namely KV Retrieval (KV), Prefix–Suffix Matching (PS), and Variable Tracking (VT). We use a frozen Ministral-3-8B-Instruct model on 90 contexts, with 5 independently evaluated requests per context. We compare full KV with StreamingLLM (Xiao et al., 2024), frozen pre-query SnapKV, query-aware SnapKV, and KIVI (Liu et al., 2024). To distinguish information that is merely retained from information that is actually used, we additionally construct 150 matched KV Retrieval pairs that differ only in the queried value and transplant the queried record's KV entries bidirectionally between their caches. Full details are given in Appendices C.4.1 and C.4.3.

Results. On Part A (Table 1), larger budgets improve retrieval. At comparable cache sizes, frozen SnapKV outperforms StreamingLLM on VT but trails on KV and PS. Four-bit KIVI maintains near-Full-KV performance using 31.1% of its physical cache size. Two-bit precision sharply reduces retrieval despite retaining every position, illustrating Remark B.14. The reported KIVI sizes already include quantization metadata and BF16 residuals (Table 5). Across 12 task–budget comparisons, query-aware SnapKV improves task scores over frozen SnapKV by 0.13–71.33 percentage points. The paired 95% confidence intervals for these gains lie entirely above zero in 11 comparisons. The gains extend Experiment 3's timing advantage and qualitatively support part (b) of Theorem 2.1. The

Table 1: SCBench retrieval performance and cache size on KV Retrieval (KV), Prefix–Suffix Matching (PS), and Variable Tracking (VT) tasks using frozen BF16 Ministral 3 8B Instruct 2512: 90 contexts, 450 queries. Values show mean accuracy (%) $\pm$ half-widths of 95% confidence intervals from paired context bootstrapping.

Method	Budget	Cache	KV	PS	VT	Preserve
Reusable pre-query cache						
Full KV	Full	100%	95.3 ± 3.7	75.3 ± 6.3	94.4 ± 2.3	100%
StreamingLLM	$3/4$	74.7%	72.7 ± 14.7	42.0 ± 6.7	40.7 ± 7.1	52.7%
	$1/2$	49.8%	46.7 ± 16.0	28.0 ± 8.3	18.7 ± 3.3	27.8%
	$1/4$	24.9%	22.0 ± 13.3	18.0 ± 7.0	17.9 ± 2.1	14.4%
	$1/8$	12.4%	12.0 ± 10.0	10.0 ± 4.3	15.9 ± 3.1	7.7%
SnapKV (frozen)	$3/4$	76.1%	70.7 ± 8.3	30.7 ± 8.7	96.1 ± 1.7	69.8%
	$1/2$	50.9%	26.0 ± 10.7	6.7 ± 4.3	90.9 ± 3.6	40.5%
	$1/4$	25.7%	6.0 ± 5.7	2.7 ± 2.3	78.4 ± 4.9	22.3%
	$1/8$	13.1%	1.3 ± 1.7	0.0 ± 0.0	65.1 ± 6.0	12.7%
KIVI	4-bit	31.1%	95.3 ± 3.7	70.0 ± 7.7	95.6 ± 1.8	96.0%
	2-bit	18.7%	17.3 ± 7.0	10.7 ± 5.0	82.9 ± 4.5	37.1%
Query-aware selection after the query						
SnapKV (query-aware)	$3/4$	75.7%	96.0 ± 2.7	67.3 ± 7.3	96.3 ± 1.5	95.9%
	$1/2$	50.5%	92.7 ± 4.3	43.3 ± 6.7	96.0 ± 1.5	83.6%
	$1/4$	25.2%	77.3 ± 7.0	12.0 ± 4.7	94.4 ± 1.9	63.6%
	$1/8$	12.6%	36.0 ± 10.3	4.7 ± 3.0	92.3 ± 2.1	42.8%

gains use Full-KV query processing and 100% reusable prefix state (Table 5), so the smaller answer-time cache illustrates query-specific compression, while Corollary B.12 concerns the complete state retained for future queries. Part B establishes causal use of retained state. Full-KV and four-bit KIVI transfers achieve approximately 95–97% donor accuracy, whereas two-bit KIVI falls to 25–26%. SnapKV donor accuracy increases with ρ, but query-aware transfers remain well below natural retrieval. Together, the interventions illustrate Theorem 2.1's interpretation of the complete cache as information passed to subsequent computation. See Appendices C.4.2 and C.4.4 for full analysis.

4 Discussions

Limitations. Our framework concerns persistent information, rather than computational efficiency. The decoder is with various assumptions: it is nonuniform in the input length, has access to the public information, and may use arbitrary transition computation. Consequently, persistent-cache complexity does not capture metrics like running time, I/O cost, energy consumption, etc. The state model is also more general than any particular Transformer implementation. A finite-precision KV cache can be represented within our model, but an abstract finite-state transition need not be realizable by a fixed Transformer architecture. Our results apply directly to the decision and retrieval problems studied in the paper. Their memory lower bounds can also constrain broader tasks containing these problems as subproblems. However, such extensions require an explicit reduction and an appropriate error criterion. Finally, the empirical study is diagnostic and limited in scale.

Conclusion. We formulated complete-cache accounting for causal decoders in the classical nonuniform streaming model. For independent uniform records, the Fano converse and rate–distortion constructions characterize memory at the database boundary under uniform and block-constant query laws, with an explicit block-restriction gap for general laws. The characterization quantifies the alphabet-dependent benefit of general summaries over literal retention, and perfect codes give explicit finite attainable points. Selection indices and repeated requests require separate accounting, while communication and state-packing arguments identify the effects of query timing and reusable randomness. Our experiments complement these results across *(i)* formal languages, *(ii)* associative recall, and *(iii)* finite KV cache settings. Together, they illustrate both the predicted memory regimes and the importance of when future queries become accessible.

AI USE STATEMENT

In this paper, we use generative AI tools, namely ChatGPT, to help with polishing our writing and narration of our theoretical findings.

REFERENCES

Scott Aaronson. Limitations of quantum advice and one-way communication. In *Proceedings. 19th IEEE Annual Conference on Computational Complexity, 2004.*, pp. 320–332. IEEE, 2004. doi: 10.1109/CCC.2004.1313854.

Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pp. 20–29, 1996. doi: 10.1145/237814.237823.

Moritz Brösamle and Stephan Eckstein. The expressive power of low precision softmax transformers with (summarized) chain-of-thought. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2605.18079. arXiv:2605.18079.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2 edition, 2006. doi: 10.1002/047174882X.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, 2022. doi: 10.52202/068431-1189. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

Yoav Gelberg, Yam Eitan, Michael Bronstein, Yarin Gal, and Haggai Maron. Training transformers for KV cache compressibility. *arXiv preprint arXiv:2605.05971*, 2026. URL https://arxiv.org/abs/2605.05971.

Themistoklis Haris and Krzysztof Onak. Compression barriers in autoregressive transformers. In *Proceedings of the Thirty-Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pp. 2757–2785. PMLR, 2025. URL https://proceedings.mlr.press/v291/haris25a.html.

Lukas Haverbeck, Carmen Amo Alonso, Andres Felipe Posada-Moreno, Sebastian Trimpe, and Marco Pavone. The risk of KV cache compression. *arXiv preprint arXiv:2607.01520*, 2026. URL https://arxiv.org/abs/2607.01520.

Rahul Jain and Ashwin Nayak. The space complexity of recognizing well-parenthesized expressions in the streaming model: The index function revisited. *IEEE Transactions on Information Theory*, 60(10):6646–6668, 2014. doi: 10.1109/TIT.2014.2339859.

Jiantong Jiang, Peiyu Yang, Rui Zhang, and Feng Liu. Towards efficient large language model serving: A survey on system-aware KV cache optimization. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 38450–38476, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1916. URL https://aclanthology.org/2026.findings-acl.1916/.

Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, Cambridge, UK, 1997. ISBN 978-0-521-56067-2. doi: 10.1017/CBO9780511574948.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV cache-centric analysis of long-context methods. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a540b17fb2295c736d5afd6c507acf66-Abstract-Conference.html.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970, 2024. doi: 10.52202/079017-0722. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026. doi: 10.48550/arXiv.2601.08584. URL https://arxiv.org/abs/2601.08584.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/liu24bz.html.

Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014. doi: 10.1137/130926122.

Ashwin Nayak. Optimal lower bounds for quantum automata and random access codes. In *40th Annual Symposium on Foundations of Computer Science*, pp. 369–376. IEEE Computer Society, 1999. doi: 10.1109/SFFCS.1999.814608.

A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958. doi: 10.1090/S0002-9939-1958-0135681-9.

Ilan Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39(2):67–71, 1991. doi: 10.1016/0020-0190(91)90157-D.

Barna Saha and Christopher Ye. I/O complexity of attention, or how optimal is FlashAttention? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43024–43042. PMLR, 2024. URL https://proceedings.mlr.press/v235/saha24a.html.

J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980. doi: 10.1145/322217.322225.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024. doi: 10.1162/tacl_a_00663. URL https://aclanthology.org/2024.tacl-1.30/.

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*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/7181-attention-is-all-you-need.

Xiao Wang. How much cache does reasoning need? depth-cache tradeoffs in KV-compressed transformers. *arXiv preprint arXiv:2604.17935*, 2026. URL https://arxiv.org/abs/2604.17935.

Ingo Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000. ISBN 978-0-89871-458-6. doi: 10.1137/1.9780898719789.

Jeffrey Willette, Heejun Lee, and Sung Ju Hwang. Delta attention: Fast and accurate sparse attention inference by delta correction. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0403.

URL https://papers.neurips.cc/paper_files/paper/2025/hash/11cc3abb01080339ceed66eff7c93920-Abstract-Conference.html.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, pp. 21875–21895, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5e5fd18f863cbe6d8ae392a93fd271c9-Abstract-Conference.html.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In *Advances in Neural Information Processing Systems*, volume 37, pp. 10202–10235, 2024. doi: 10.52202/079017-0327. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/13d7f172259b11b230cc5da8768abc5f-Abstract-Conference.html.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710, 2023. doi: 10.52202/075280-1506. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

A Related Work

KV-cache compression and long-context evaluation. A large body of literature studies how to reduce the memory and computational costs of long-context inference. StreamingLLM (Xiao et al., 2024) retains attention sinks and a rolling window, while H2O (Zhang et al., 2023) prioritizes recent tokens and heavy hitters. SnapKV (Li et al., 2024) uses an observation window to identify positions that are likely to matter during generation, whereas Delta Attention (Willette et al., 2025) corrects the distribution shift induced by sparse attention. SCBench (Li et al., 2025) evaluates long-context methods specifically from the perspective of KV-cache reuse and highlights difficulties on exact retrieval and multi-request workloads. These works primarily ask which cache performs well for a given model and workload, and we differ by finding how many distinguishable persistent states are generally required by the underlying task. Thus, our results are not a competing algorithm for cache compression, and our experiments should not be regarded as a replacement for benchmarks like SCBench.

Lower bounds and finite-precision theory. Haris & Onak (2025) establish communication-based space lower bounds for online attention-based token generation. Our object is different: we obtain task-dependent bounds on persistent information, rather than modeling dot-product attention or an nd-scale attention lower bound. Meanwhile, Brösamle & Eckstein (2026) explore the expressiveness of low-precision softmax decoders with ordinary and summarized Chain-of-Thought. Their focus is on which computations the architecture can realize, while ours is on minimizing the persistent state necessary for a task.

Recent theory also studies resources and representation choices that our model leaves unrestricted. Wang (2026) analyzes pointer chasing under a bounded shared KV cache and a locality-respecting controller, proving depth lower bounds while leaving a stronger product bound conjectural. Gelberg et al. (2026) show that the same sequence-to-vector function can admit Transformer implementations with different KV compressibility. Haverbeck et al. (2026) characterize minimax attention-approximation risk through intrinsic cache compressibility. Our bounds minimize persistent state over unrestricted causal transitions for a specified task and error criterion. They therefore provide an information-accounting synthesis, rather than improvements to those depth bounds or approximation guarantees for a fixed attention representation.

Efficient attention and I/O complexity. Saha & Ye (2024) use I/O complexity to characterize data movement for exact attention and the regimes in which FlashAttention-style algorithms (Dao et al., 2022) are optimal. I/O complexity concerns how attention is evaluated, while persistent-cache complexity concerns how many prefix-dependent states must remain distinguishable. Neither

measure subsumes the other, as a decoder can have low persistent-cache complexity but expensive transitions, and vice versa.

Formal languages and communication complexity. Our mathematical framework connects naturally to several classical models. For deterministic coding, future equivalence is closely related to Myhill–Nerode theory (Nerode, 1958). The deterministic decoder can also be viewed as a length-n ordered layered branching program whose layer width is the number of cache states (Wegener, 2000). This differs from work asking which formal languages particular Transformer architectures can recognize (Strobl et al., 2024, Yang et al., 2024). Our model, instead, has an explicit persistent state, so every regular language is with constant cache by DFA simulation. The streaming-to-communication route is classical, including the frequency-moment lower bounds of Alon et al. (1996) and the general framework of Kushilevitz & Nisan (1997). Our model is a nonuniform streaming-space measure with explicit conventions for public position and persistent randomness. The associative-recall converse uses the classical INDEX/random-access information argument (Nayak, 1999). The equality calibration uses polynomial fingerprinting (Schwartz, 1980), and the Dyck-(2) example inherits known one-pass streaming upper and lower bounds (Magniez et al., 2014, Jain & Nayak, 2014).

Rate–distortion and randomness conventions. The function f_q is the classical rate–distortion function of a uniform q-ary source under Hamming distortion (Cover & Thomas, 2006). Our achievability statements translate its block codes into deterministic cache indices and combine them with the query-weighted converse. The blockwise extension tracks varying distortion targets and fixes coordinate permutations using the query law. The private/public equality separation is also classical (Kushilevitz & Nisan, 1997), and Newman's theorem relates public and private communication with logarithmic overhead (Newman, 1991). We include a direct cache-state packing proof to make clear which reusable randomness must be charged. These are applications and refinements of established tools for the cache accounting problem, rather than new source-coding or communication-separation principles.

B Detailed proof of theoretical findings

B.1 Proof of the transfer part of Theorem 2.1

Proof. Fix n and t, and let D be an ε-error causal decoder such that $|\mathcal{C}_{n,t}| \le 2^s$. Consider the one-way communication problem

$$F_{\mathcal{T},n,t}(u,z) = T_n(uz) \qquad u \in \Sigma^t, \quad z \in \Sigma^{n-t}.$$

We construct a one-way protocol between Alice and Bob. Alice receives the prefix $u = u_1 \cdots u_t \in \Sigma^t$, while Bob receives the suffix $z = z_1 \cdots z_{n-t} \in \Sigma^{n-t}$.

Alice begins from the public initial cache state $C_{n,0} \in \mathcal{C}_{n,0}$, setting $C_0 = C_{n,0}$. She then simulates the first t transitions of the decoder on u. In the deterministic case, the state evolves as $C_j = U_{n,j}(C_{j-1}, u_j),\ j = 1, \ldots, t$. In the randomized case, she samples from the corresponding transition kernels. After processing the entire prefix, Alice obtains a realized cache state $C_t \in \mathcal{C}_{n,t}$. Under the private-state convention, any value generated during the prefix that will be needed later must already be represented in C_t. Thus, C_t contains the complete persistent information produced by the prefix computation.

Alice then sends the cache state. By assumption, $|\mathcal{C}_{n,t}| \le 2^s$. Thus, the possible states at time t can be assigned distinct binary representations of length at most s. Let $\mathrm{enc}_{n,t} : \mathcal{C}_{n,t} \to \{0,1\}^s$ be any injective encoding, padding shorter representations if necessary. Alice sends $M = \mathrm{enc}_{n,t}(C_t)$ to Bob. Hence, the communication cost is at most s bits, and no description of n or t is required because they are public parameters.

Upon receiving M, Bob reconstructs $C_t = \mathrm{enc}_{n,t}^{-1}(M)$. Then, he processes his suffix z using the remaining decoder transitions: $C_{t+j} \sim U_{n,t+j}(C_{t+j-1}, z_j),\ j = 1, \ldots, n-t$. After the final suffix symbol, Bob reaches $C_n \in \mathcal{C}_{n,n}$ and outputs $A_n(C_n) \in \mathcal{Y}_{\mathcal{T}}$, aiming to compute $T_n(uz)$.

Importantly, C_t is the entire persistent interface between the processed prefix and all future computation. Once C_t is fixed, later transitions cannot access the original prefix u except through information encoded in C_t.

Consider first the private-state convention. Conditional on $C_t = c$, Bob starts the suffix computation from exactly the same state c from which an uninterrupted execution would continue after processing u. Any prefix randomness that needs to influence later computation is already in c. Any randomness generated after t is fresh and is sampled by Bob according to exactly the same transition kernels as in the uninterrupted decoder.

Therefore, for every prefix state c of positive probability and every final state $c' \in \mathcal{C}_{n,n}$, we have

$$\Pr_{\Pi}[C_n = c' | C_t = c, z] = \Pr_{D}[C_n = c' | C_t = c, z],$$

with Π being the communication protocol. Applying the deterministic output map A_n, we obtain, for every $y \in \mathcal{Y}_{\mathcal{T}}$,

$$\Pr_{\Pi}[\Pi(u, z) = y | C_t = c] = \Pr_{D}[D(uz) = y | C_t = c].$$

Averaging over the random prefix state C_t gives

$$\Pr_{\Pi}[\Pi(u, z) = y] = \Pr_{D}[D(uz) = y]. \tag{6}$$

This equality holds for every $y \in \mathcal{Y}_{\mathcal{T}}$.

Thus, the protocol reproduces the decoder's entire output distribution, not merely its probability of a binary acceptance event.

Since D has worst-case error at most ε,

$$\Pr_{D}[D(uz) \neq T_n(uz)] \leq \varepsilon.$$

Hence,

$$\Pr_{\Pi}[\Pi(u, z) \neq F_{\mathcal{T},n,t}(u, z)] = \Pr_{\Pi}[\Pi(u, z) \neq T_n(uz)] \leq \varepsilon.$$

This holds for every $u \in \Sigma^t,\ z \in \Sigma^{n-t}$. Therefore, Π is an ε-error one-way protocol using at most s communicated bits.

If the decoder instead operates under the public-coin convention, Alice and Bob have access to the same public random tape. Alice simulates the prefix using that tape and sends the resulting cache state. Bob resumes the computation using the same public tape together with the transmitted state. Conditional on the public tape, the preceding simulation argument is identical. Therefore, the protocol again reproduces the decoder's output distribution and has error at most ε. Thus, the communication protocol preserves the randomness convention. □

B.2 Proof of part (b) of Theorem 2.1

Proof. For a communication function $f(u, z)$, let $r(f)$ count the distinct functions $z \mapsto f(u, z)$ as the sender's input u varies. For Boolean outputs, Aaronson (2004)'s characterization gives the minimum deterministic one-way message length as $D^{\rightarrow}(f) = \lceil \log_2 r(f) \rceil$. The same argument holds for $[q]$-valued outputs: different answer functions require different messages, while their identifiers suffice. By Theorem 2.1, these communication lower bounds also bound the cache at the corresponding prefix–suffix split.

When the prefix is X, its answer function is $i \mapsto X_i$. Distinct databases differ at some coordinate, so there are q^m such functions. When the prefix is i, its answer function is $X \mapsto X_i$. Any two distinct indices can be separated by a database assigning them different values, giving m functions. After both blocks, the suffix is empty, so prefixes are distinguished only by the requested value X_i, giving q functions. Taking logarithms yields the two bounds and the stated $\lceil \log_2 q \rceil$ bound after both blocks.

The bounds are attained by storing, respectively, an encoding of X, the index i, or the answer X_i at the indicated boundary. These states can be computed causally using unrestricted memory before that boundary, and each suffices for the remaining computation. □

B.3 Exact deterministic characterization and proof

Remark B.1 (Streaming space and branching-program width). The quantity $K_{\mathcal{T}}^{\det}(n)$ is precisely the minimum persistent space of a length-nonuniform one-pass streaming computation with public position and unrestricted transition computation. Equivalently, it is the ceiling of the logarithm of the minimum maximum layer width of an ordered layered branching program for T_n, with variables read in input order (Wegener, 2000). The randomized variants are the corresponding streaming-space measures under the specified randomness conventions. We use this established model to account for the entire surviving state of a cache, including reusable randomness. The recall results in Section 2.2 combine this accounting with classical rate–distortion theory to characterize uniform and block-constant query laws and to bound the cost of blockwise allocation for general laws.

For $u, u' \in \Sigma^t$, write $u \equiv_{\mathcal{T},n,t} u'$ if $T_n(uz) = T_n(u'z)$ for every $z \in \Sigma^{n-t}$. Let $N_{\mathcal{T}}(n,t)$ be the number of these future-equivalence classes and $N_{\mathcal{T}}(n) = \max_t N_{\mathcal{T}}(n,t)$.

Corollary B.2 (Exact finite-length cache characterization)**.** *For every finite-valued task family $\mathcal{T}$ and every $n \geq 0$,*

$$K_{\mathcal{T}}^{\det}(n) = \max_{0 \leq t \leq n} \lceil \log_2 r(F_{\mathcal{T},n,t}) \rceil = \lceil \log_2 N_{\mathcal{T}}(n) \rceil . \tag{7}$$

Here $r(F)$ is the number of distinct rows of the communication matrix of F, and $r(F_{\mathcal{T},n,t}) = N_{\mathcal{T}}(n,t)$. The lower bound follows from Theorem 2.1 and the classical identity $D^{\rightarrow}(F) = \lceil \log_2 r(F) \rceil$ (Aaronson, 2004). The matching class-encoding construction is given in Appendix B.3.

Proof. For each split t, two prefixes index the same row of $F_{\mathcal{T},n,t}$ precisely when they are future-equivalent. Thus, $r(F_{\mathcal{T},n,t}) = N_{\mathcal{T}}(n,t)$. By Theorem 2.1 and the deterministic distinct-row characterization (Aaronson, 2004), any exact decoder uses at least $\lceil \log_2 N_{\mathcal{T}}(n,t) \rceil$ bits at that split. Maximizing over t gives the lower bound.

For the upper bound, set $k = \lceil \log_2 N_{\mathcal{T}}(n) \rceil$ and choose, at each layer, an injective map

$$\mathrm{code}_{n,t} : \Sigma^t / \equiv_{\mathcal{T},n,t} \longrightarrow \{0,1\}^k. \tag{8}$$

Use its image as the state set and initialize at the code of the empty prefix. If $t < n$ and $u \equiv_{\mathcal{T},n,t} u'$, then $T_n(uaw) = T_n(u'aw)$ for every $a \in \Sigma$ and $w \in \Sigma^{n-t-1}$. Hence, $ua \equiv_{\mathcal{T},n,t+1} u'a$, and the update

$$U_{n,t+1}\big(\mathrm{code}_{n,t}([u]), a\big) = \mathrm{code}_{n,t+1}([ua]) \tag{9}$$

is well defined. At $t = n$, the class $[x]$ determines $T_n(x)$, so the final output map is also well defined. Every layer uses at most k bits, including the case $k = 0$. This proves (7). □

B.4 Full proof of Proposition B.3

Proposition B.3 (Language calibrations)**.**
1) Regular languages. *If the language L can be recognized by a Deterministic Finite Automaton (DFA) with state set Q, then*

$$K_L^{\det}(n) \leq \lceil \log_2 |Q| \rceil. \tag{10}$$

2) Equality. *For $m \geq 1$, recognizing $\mathrm{EQ}_m = \{x\#y : x, y \in \{0,1\}^m,\ x = y\}$ exactly and deterministically requires at least m bits of persistent state after reading x. On this length-$(2m+1)$ slice, for $0 < \delta < 1/2$, the private-state convention admits a one-sided-error polynomial-fingerprinting upper bound of $O(\log(m/\delta))$ persistent bits. For inverse-polynomial δ, this becomes $O(\log m)$. Proposition 2.5 gives a matching private-state lower bound for fixed error and separates this convention from public coins.*

3) Dyck-(2). *For every fixed constant $\varepsilon \in (0, 1/2)$ and sufficiently large even n, the published one-pass streaming bounds imply*

$$\Omega(\sqrt{n}) \leq K_{\mathrm{Dyck}(2)}^{\mathrm{priv},\varepsilon}(n) \leq O(\sqrt{n \log n}), \tag{11}$$

and, hence, $K_{\mathrm{Dyck}(2)}^{\mathrm{priv},\varepsilon}(n) = \widetilde{\Theta}(\sqrt{n})$ (Magniez et al., 2014, Jain & Nayak, 2014).
The constant-error lower bound is due to Jain & Nayak (2014). The stronger $\Omega(\sqrt{n \log n})$ bound of

Magniez et al. (2014) is stated for one-pass algorithms with two-sided error $O(1/(n \log n))$, where n is the input length. It does not establish that rate at fixed error. Detailed proofs are provided in Appendix B.4.

Proof. **Regular languages.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for L. Store the current state q_t. Initially, q_0 is stored, and on symbol x_t, we update $q_t = \delta(q_{t-1}, x_t)$. Induction on t shows that the cached state is exactly the state reached by M on the first t symbols. At time n, the output is 1 precisely when $q_n \in F$. Hence, the decoder recognizes L exactly.

There are at most $|Q|$ cache states at every layer. The same construction works for every n, and thus

$$K_L^{\det}(n) \le \lceil \log_2 |Q| \rceil.$$

Equality. After the first block, distinct $x, x' \in \{0,1\}^m$ must reach distinct states, in that suffix $\#x$ accepts after x and rejects after x'. Hence, at least 2^m states (or m bits) are necessary.

For $0 < \delta < 1/2$, pick a prime $q > 1 + m/\delta$ and sample α uniformly from $\mathbb{F}_q^\times$. The sampled α persists and is counted. Sampling occurs in the first transition, preserving the fixed initial state. While reading x and y, we maintain

$$d = \sum_{j=1}^{m} (x_j - y_j)\alpha^j \pmod q.$$

The block and exponent are identified as public information, so the state need only retain α and d, as well as constant control. A validity flag checks the binary blocks and the delimiter at their prescribed public positions and rejects every malformed string. If $x = y$, then $d = 0$, $\forall \alpha$. If $x \neq y$, the difference polynomial is nonzero and with degree at most m, so it has at most m roots in $\mathbb{F}_q$ (Schwartz, 1980). Consequently, $\Pr[d = 0] \le m/(q-1) < \delta$. Bertrand's postulate supplies a prime $q = O(m/\delta)$, so the persistent state uses $O(\log(m/\delta))$ bits.

For comparison, if independent length-m public random strings are freely and consistently accessible at both blocks, one can maintain repeated random inner products. Each repetition has collision probability of $1/2$, and $\lceil \log_2(1/\delta) \rceil$ independent repetitions have total collision probability at most δ. Thus, $O(\log(1/\delta))$ accumulator bits, together with a validity flag, suffice. This is why the public-coin and private-state conventions must remain distinct.

Dyck-(2). Let $\Sigma_D = \{(,), [,]\}$ denote the four-symbol parenthesis alphabet. A string belongs to Dyck-(2) precisely when it is correctly nested, and each closing parenthesis matches the type of its opening parenthesis. This means that if n is odd, the decoder with public n may use a singleton state set at every layer and output zero. Hence, $K^{\mathrm{priv},\varepsilon}_{\mathrm{Dyck}(2)}(n) = 0$ if n is odd. What we aim to prove becomes: for every fixed $0 < \varepsilon < 1/2$, there exist positive constants $g_\varepsilon, G_\varepsilon$ such that for every sufficiently large even n, we have

$$g_\varepsilon \sqrt{n} \le K^{\mathrm{priv},\varepsilon}_{\mathrm{Dyck}(2)}(n) \le G_\varepsilon \sqrt{n \log n}.$$

The one-pass upper bound of Magniez et al. (2014) provides a randomized recognizer using $O(\sqrt{n \log n})$ bits. It accepts members with certainty, and has false acceptance probability at most n^{-c} for any fixed $c > 0$. For the lower bound, we use Jain & Nayak (2014): the hard-input simulation for parenthesis streams of length at most $4r^2$, with handoff messages of at most b bits in one pass, yields a two-party Augmented Index protocol with information costs at most b and b/r. At fixed error below $1/4$, their information-cost trade-off forces $b = \Omega(r)$ for sufficiently large even r. We apply this communication reduction directly to cache states, so unrestricted transition computation and length-dependent transition tables cause no additional assumption.

Take the recognizer from Magniez et al. (2014) with $c = 1$. At every token boundary, retain its complete configuration. The algorithm uses a hashing modulus and seed with polynomial size, which takes $O(\log n)$ bits to retain. If the full configuration can be serialized in $B(n) = O(\sqrt{n \log n})$ bits, there are at most $2^{B(n)}$ such configurations. A transition of the cache decoder simulates all computation of the recognizer until the next input boundary and retains only the resulting configuration. The fixed initial state is preserved by incorporating any random initialization into the first transition. The final transition may likewise incorporate postprocessing, leaving a deterministic output map.

This simulation preserves the output distribution. For every sufficiently large n, $n^{-1} \le \varepsilon$, so its error is at most ε. Its complete persistent-bit count is $B(n)$, meaning that

$$K^{\mathrm{priv},\varepsilon}_{\mathrm{Dyck}(2)}(n) = O(\sqrt{n \log n}).$$

The finitely many smaller lengths can be handled by exact storage without changing the asymptotic statement.

Consider any cache decoder D on a fixed even length n, and write $S = \max_{0 \le t \le n} S_D(n,t)$. First, we reduce its error to a constant at most $1/8$ within the error regime of the information-cost bound in Jain & Nayak (2014). Let $\eta = 1/2 - \varepsilon > 0$. Run an odd number a of independent copies in parallel and take a majority vote, choosing a to be the least odd integer with $a \ge 2/\eta^2$. For any fixed input, let E_j indicate that copy j is wrong. For $\bar{E} = a^{-1} \sum_{j=1}^{a} E_j$, independence gives

$$\mathbb{E}[\bar{E}] \le \varepsilon, \qquad \mathrm{Var}(\bar{E}) \le \frac{1}{4a}.$$

A majority error requires $\bar{E} - \mathbb{E}[\bar{E}] \ge \eta$. Therefore, Chebyshev's inequality gives

$$\begin{aligned} \Pr[\text{majority error}] &\le \Pr[|\bar{E} - \mathbb{E}[\bar{E}]| \ge \eta] \\ &\le \frac{\mathrm{Var}(\bar{E})}{\eta^2} \\ &\le \frac{1}{4a\eta^2} \le \frac{1}{8}. \end{aligned}$$

The joint cache has at most 2^{aS} states at each layer. The factor a depends only on the fixed ε, not on n.

Choose $r = 2\lfloor \sqrt{n}/4 \rfloor$, so r is even, $r = \Theta(\sqrt{n})$, and $4r^2 \le n$. For any even-length hard input w with $|w| \le 4r^2 \le n$, append a fixed balanced suffix to obtain

$$\mathrm{pad}_n(w) = w\,(())^{(n-|w|)/2}.$$

This transformation preserves membership. If w is balanced and well-typed, concatenating balanced pairs preserves that property. A prefix underflow or a type mismatch in w cannot be repaired by appending symbols. If w ends with positive height, appending balanced pairs leaves that height unchanged. Therefore,

$$w \in \mathrm{Dyck}(2) \iff \mathrm{pad}_n(w) \in \mathrm{Dyck}(2).$$

Apply the published hard-input handoff construction to the amplified decoder on $\mathrm{pad}_n(w)$. At a handoff, transmit the joint cache, together with the number of symbols already simulated. The latter costs at most $\lceil \log_2(n+1) \rceil$ bits. It makes the current layer explicit, even when a segment length depends on a player's input. The final simulator knows $|w|$ from this counter and can process the appended balanced suffix locally. Thus, every handoff size has at most

$$b = aS + \lceil \log_2(n+1) \rceil + O(1).$$

The augmented-index reduction and trade-off stated above now apply to this handoff protocol with error at most $1/8$. They imply $b \ge g_0 r$ for a positive constant g_0. Consequently, for a constant $C_0 \ge 0$ independent of n,

$$aS + \lceil \log_2(n+1) \rceil + C_0 \ge 2g_0 \lfloor \sqrt{n}/4 \rfloor.$$

Since $\log n = o(\sqrt{n})$, and a is a constant for fixed ε, this yields $S = \Omega_\varepsilon(\sqrt{n})$. The above inequality holds for every permitted D, so minimizing over decoders leads to $\Omega_\varepsilon(\sqrt{n}) \le K^{\mathrm{priv},\varepsilon}_{\mathrm{Dyck}(2)}(n)$. Together with the upper bound, it gives

$$K^{\mathrm{priv},\varepsilon}_{\mathrm{Dyck}(2)}(n) = \tilde{\Theta}(\sqrt{n}),$$

among even n. This completes our proposition. □

B.5 THE PRIVATE-STATE EQUALITY LOWER BOUND

Proof of Proposition 2.5. The private-state fingerprint and public-coin inner-product constructions in Appendix B.4 give the upper bounds, including rejection of malformed inputs. For the lower bound, consider a private-state decoder with at most $N \le 2^S$ states immediately after reading $x \in \{0,1\}^m$. Let $p_x \in \mathbb{R}^N$ be the probability vector of this state. For the suffix $\#x$, let $b_x(c) \in [0,1]$ be the acceptance probability when the continuation starts from state c. All surviving randomness is in that state, so the same vector b_x describes the continuation after any prefix. Worst-case error at most δ implies, for $x' \neq x$,

$$\langle p_x, b_x \rangle \ge 1-\delta, \qquad \langle p_{x'}, b_x \rangle \le \delta, \qquad \|p_x - p_{x'}\|_1 \ge 2(1-2\delta). \tag{12}$$

The last inequality uses $\sup_{b\in[0,1]^N} |\langle p-p', b\rangle| = \|p-p'\|_1/2$ for probability vectors.

Set $r = 1-2\delta > 0$. The open ℓ_1 balls of radius r centered at the 2^m vectors p_x are disjoint. Each lies in the radius-$(1+r)$ ball about the origin because $\|p_x\|_1 = 1$. Comparing their N-dimensional volumes gives

$$2^m r^N \le (1+r)^N, \qquad m \le N\log_2(1+1/r) \le 2^S \log_2(1+1/r). \tag{13}$$

Taking logarithms yields $S \ge \log_2 m - \log_2 \log_2\Big(1+\frac{1}{1-2\delta}\Big)$. For fixed δ, this and the fingerprint upper bound give $\Theta_\delta(\log m)$, while the public-coin construction uses $\lceil \log_2(1/\delta)\rceil$ parity accumulators and a constant number of control bits. A free reusable tape invalidates the state-only factorization through b_x, which is precisely the distinction between the two conventions. □

B.6 PROOF OF THE CONVERSE PARTS OF THEOREM 2.2

Proof. Let Z_i denote any fresh randomness used to decode coordinate i. By definition, Z_i is independent of (X, C). A persistent seed would already be part of C. Set $E_i = \mathbf{1}[\widehat{X}_i \neq X_i]$. Given (C, Z_i), the error indicator is determined by X_i, and on an error there are at most $q-1$ possible values of X_i. The entropy chain rule therefore gives the q-ary Fano inequality for every $e_i \in [0,1]$:

$$\begin{aligned} H(X_i \mid C, Z_i) &= H(E_i \mid C, Z_i) + H(X_i \mid E_i, C, Z_i) \\ &\le h_2(e_i) + e_i \log_2(q-1). \end{aligned}$$

Independence of Z_i implies

$$H(X_i \mid C, Z_i) = H(X_i \mid C).$$

Because $(X_1, \dots, X_m)$ are independent and uniform, we have $H(X) = m\log_2 q$. Moreover,

$$\begin{aligned} H(X \mid C) &= \sum_{i=1}^m H(X_i \mid X_1, \dots, X_{i-1}, C) \\ &\le \sum_{i=1}^m H(X_i \mid C) \\ &\le \sum_{i=1}^m \left[h_2(e_i) + e_i \log_2(q-1)\right]. \end{aligned}$$

The cache has at most 2^S values, so $S \ge H(C) \ge I(X:C)$. Combining the above formulas yields

$$\begin{aligned} S &\ge H(X) - H(X \mid C) \\ &\ge \sum_{i=1}^m \left[\log_2 q - h_2(e_i) - e_i \log_2(q-1)\right]. \end{aligned}$$

When $q = 2$, the final logarithmic term vanishes. For an integer S, taking a ceiling of this final sum gives the integer version. No coordinatewise rounding follows from the entropy inequalities.

A deterministic MAP decision attains $e_i^\star$, since both alphabets are finite. Conditional on each cache value, randomizing the prediction cannot improve on a most probable symbol. Thus, $e_i^\star \le e_i$, and

ignoring the cache and predicting any fixed symbol gives $e_i^\star \le 1 - 1/q$. Applying the same Fano argument to the MAP decisions gives $S \ge S_{\mathrm{Fano}}(e^\star)$. The derivative of $f_q(e) = \log_2 q - h_2(e) - e\log_2(q-1)$ is

$$f_q'(e) = \log_2\left(\frac{e}{(1-e)(q-1)}\right) \le 0, \qquad 0 < e \le 1 - 1/q.$$

Continuity at zero then proves $S \ge S_{\mathrm{Fano}}(u)$ whenever $e_i^\star \le u_i \le 1 - 1/q$ for every coordinate.

For part (b), the MAP errors satisfy $0 \le e_i^\star \le c_q$ and $\sum_i \pi_i e_i^\star \le \sum_i \pi_i e_i \le \bar{e}$. They are therefore feasible in (3). Minimizing the bound in part (a) proves the converse. The feasible set is nonempty and compact, so its minimum exists. Under uniform queries, Jensen's inequality and the monotonicity of f_q on $[0, c_q]$ give

$$\sum_{i=1}^m f_q(a_i) \ge m f_q\left(\frac{1}{m}\sum_{i=1}^m a_i\right) \ge m f_q(\bar{e}). \tag{14}$$

The constant vector $a_i = \bar{e}$ attains equality in the variational problem.

For positive query masses and $0 < \bar{e} < c_q$, stationarity takes the form $f_q'(a_i) + \lambda\pi_i = 0$, which gives (4). Its weighted average is continuous and strictly decreasing in λ, from c_q at zero to zero at infinity. Thus, exactly one positive multiplier satisfies the error constraint. For its candidate $a^\star$ and any feasible vector v, convexity gives

$$\sum_i f_q(v_i) \ge \sum_i f_q(a_i^\star) - \lambda \sum_i \pi_i (v_i - a_i^\star) \ge \sum_i f_q(a_i^\star). \tag{15}$$

Since $f_q''(a) = 1/[a(1-a)\ln 2] > 0$ in the interior, strict convexity on $[0, c_q]$ proves uniqueness. A coordinate with $\pi_i = 0$ minimizes its own term at $a_i = c_q$, after which the same argument applies to positive masses. At the endpoints, $\mathcal{R}_\pi(0) = |\{i : \pi_i > 0\}| \log_2 q$ and $\mathcal{R}_\pi(c_q) = 0$.

□

Remark B.4 (General database laws). Let X have any law on $[q]^m$, and let $C = \psi(X, U)$ with U independent of X contain all persistent randomness and satisfy $|\operatorname{supp} C| \le 2^S$. Define the coordinate Bayes errors $e_i^\star = \min_{\phi:\operatorname{supp}(C)\to[q]} \Pr[\phi(C) \ne X_i]$. Then,

$$S \ge \left[H(X) - \sum_{i=1}^m \left(h_2(e_i^\star) + e_i^\star \log_2(q-1)\right)\right]_+ . \tag{16}$$

Indeed, $S \ge I(X;C) = H(X) - H(X \mid C)$, conditional subadditivity gives $H(X \mid C) \le \sum_i H(X_i \mid C)$, and coordinatewise Fano bounds each summand. Independence is not needed for these steps. The uniform product law maximizes $H(X)$ and hence gives the strongest form of this entropy converse at a fixed error vector. Correlation can reduce the required state, as when every coordinate equals one uniform symbol and exact recall uses only $\lceil \log_2 q \rceil$ bits. No general-source achievability assertion is made here.

B.7 Achievability for uniform and block query laws

Definition B.5 (Memory at the database boundary)**.** For fixed q, a query law π on $[m]$, and $0 \le \bar{e} \le c_q$, define

$$S_\pi^\star(m, \bar{e}) = \min\left\{\lceil \log_2 |\operatorname{supp} C| \rceil : \sum_{i=1}^m \pi_i \Pr[\phi_i(C, Z_i) \ne X_i] \le \bar{e}\right\}. \tag{17}$$

The minimum ranges over caches $C = \psi(X, U)$ with X uniform on $[q]^m$ and encoder randomness U independent of X, and over coordinate decoders satisfying the hypotheses of Theorem 2.2. Every realized random value reused after compression is included in C. This representation permits any joint law of (X, C) with the stated uniform X-marginal, while the achievability constructions use deterministic caches $C = \psi(X)$. The encoder may inspect the full database before producing C. This measures memory at the compression boundary. It does not bound the largest state used while reading the database, which is the quantity minimized by $K_{\mathcal{T}}$.

Theorem B.6 (Block-constant query laws)**.** *Fix integers $q \geq 2$ and $g \geq 1$, and fix $\bar{e} \in (0, c_q)$. Consider a sequence of instances indexed by m, with $[m]$ partitioned into g blocks G_j of sizes $m_j \to \infty$ and $\pi = \pi^{(m)}$ constant and positive on each block. Here $\mathcal{R}_\pi$ depends on m through the number of coordinates and $\pi^{(m)}$. Then*

$$\mathcal{R}_\pi(\bar{e}) \leq S^\star_\pi(m, \bar{e}) \leq \mathcal{R}_\pi(\bar{e}) + o(m). \tag{18}$$

In particular, for every $\gamma > 0$ and all sufficiently large m, the upper bound is at most $\mathcal{R}_\pi(\bar{e}) + \gamma m + g$. Concatenating g deterministic block indices achieves it. The proof in Appendix B.7 is uniform in the block error targets, including targets approaching zero or c_q.

Corollary B.7 (Block restriction for general query laws)**.** *Under the same fixed-q, fixed-g, and growing-block assumptions, let π be any query law. Set $w_j = \pi(G_j)$ and define*

$$\mathcal{R}^{(G)}_\pi(\bar{e}) = \min_{\substack{0 \leq d_j \leq c_q \\ \sum_j w_j d_j \leq \bar{e}}} \sum_{j=1}^{g} m_j f_q(d_j). \tag{19}$$

Then

$$\mathcal{R}_\pi(\bar{e}) \leq S^\star_\pi(m, \bar{e}) \leq \mathcal{R}^{(G)}_\pi(\bar{e}) + o(m), \qquad \mathcal{R}_\pi(\bar{e}) \leq \mathcal{R}^{(G)}_\pi(\bar{e}). \tag{20}$$

A fixed permutation of each block code can be chosen using π so that its weighted error is at most $w_j d_j$. Thus, no random seed or data-dependent permutation is omitted from the cache count. The proof is in Appendix B.7.

Proof of part (c) of Theorem 2.2. The lower bound is the uniform-query case of Theorem 2.2. For the upper bound, the rate–distortion function of a uniform $[q]$ source under Hamming distortion is $f_q(D)$ for $0 \leq D \leq c_q$ (Cover & Thomas, 2006). To see the single-letter identity, Fano gives the lower bound on mutual information. Equality is attained by a uniform reproduction symbol $\widehat{X}$ and the backward channel with probabilities $\Pr[X = \widehat{X} \mid \widehat{X}] = 1 - D$ and $D/(q-1)$ for each other symbol. Its input marginal is uniform and its mutual information is $f_q(D)$.

Choose $D \in (0, \bar{e})$ with $f_q(D) < f_q(\bar{e}) + \gamma/4$. The rate–distortion coding theorem supplies, for all sufficiently large m, a deterministic encoder and a codebook $\mathcal{K}_m \subseteq [q]^m$ with

$$\log_2 |\mathcal{K}_m| \leq m\big(f_q(D) + \gamma/4\big), \qquad \frac{1}{m}\mathbb{E}\, d_H(X, \widehat{X}(X)) \leq \bar{e}. \tag{21}$$

Here d_H counts differing coordinates. The strict distortion slack permits the coding theorem's vanishing excess distortion. Store only the index of $\widehat{X}(X)$ and answer coordinate i with its ith symbol. The index requires at most $m f_q(\bar{e}) + \gamma m/2 + 1$ bits. For $m \geq 2/\gamma$, this is at most the asserted upper bound. The codebook is fixed independently of X, and no realized randomness survives. □

Lemma B.8 (Uniformity over distortion targets)**.** *For a fixed integer $q \geq 2$, there is a sequence $\eta_n \to 0$ such that, for every $d \in [0, c_q]$, a deterministic code for a uniform $[q]^n$ source has expected normalized Hamming distortion at most d and an integer index length at most $n f_q(d) + n\eta_n + 1$.*

Proof. Fix $\xi > 0$. By uniform continuity, choose a finite grid $0 = d_0 < d_1 < \cdots < d_k = c_q$ so that $f_q(d_j) - f_q(d) \leq \xi/2$ whenever $d_j \leq d \leq d_{j+1}$. At each interior grid point, the coding argument above gives distortion at most d_j and log codebook size at most $n(f_q(d_j) + \xi/2)$ for all sufficiently large n. Exact storage handles $d_0 = 0$, and a single constant reproduction handles $d_k = c_q$. There are finitely many grid points, so one length threshold works for all of them. For an arbitrary target d, use the code at the largest grid point at most d. Its distortion is at most d and its log codebook size is at most $n(f_q(d) + \xi)$. Taking a ceiling adds at most one bit. Since this holds for every $\xi > 0$ with a length threshold independent of d, a diagonal choice of thresholds yields $\eta_n \to 0$. □

Proof of Theorem B.6. The converse is Theorem 2.2. For block-constant π, the optimizer in (4) is constant on each G_j. Write its value as $d^\star_j$ and its block mass as w_j. Then $\sum_j w_j d^\star_j = \bar{e}$ and $\mathcal{R}_\pi(\bar{e}) = \sum_j m_j f_q(d^\star_j)$. Apply Lemma B.8 separately to each block and concatenate the fixed-length indices. The expected error under π is at most $\sum_j w_j d^\star_j$, and the total number of stored bits is at most

$$\sum_{j=1}^{g} \big(m_j f_q(d^\star_j) + m_j \eta_{m_j} + 1\big) \leq \mathcal{R}_\pi(\bar{e}) + m \max_j \eta_{m_j} + g. \tag{22}$$

Since g is fixed and every $m_j \to \infty$, the last two terms are $o(m)$. The uniformity of the lemma allows both π and the targets $d_j^\star$ to vary with m. □

Proof of Corollary B.7. Let (d_j) minimize (19) and choose the block codes from Lemma B.8. For a particular block, denote the code's coordinate errors by $b_1, \dots, b_{m_j}$, with $m_j^{-1} \sum_k b_k \le d_j$. They need not be equal. Average the weighted error over all cyclic permutations of the block coordinates. Uniformity of the database implies that the error vector is permuted in the same way, so this average is

$$\frac{w_j}{m_j} \sum_{k=1}^{m_j} b_k \le w_j d_j. \tag{23}$$

At least one cyclic permutation therefore achieves weighted error at most $w_j d_j$. Fix such a permutation as part of the code using the known law π. Its choice is independent of the realized database and is public, so no random seed or selection index is stored. Concatenating the permuted block codes gives the upper bound with the same $o(m)$ overhead as (22). Restricting the feasible vectors in (3) to block-constant vectors gives $\mathcal{R}_\pi \le \mathcal{R}_\pi^{(G)}$, and the unrestricted converse gives the remaining inequality. □

B.8 Literal retention and the alphabet-dependent gap

Corollary B.9 (Literal retention versus optimal summaries)**.** *Fix an integer $q \ge 2$ and $\rho \in (0,1)$, and let $B_m = \lfloor \rho m \log_2 q \rfloor$. Under uniform queries, the minimum population error among all summaries of at most B_m bits converges to*

$$D_q(\rho) = f_q^{-1}(\rho \log_2 q), \tag{24}$$

where the inverse is taken on $[0, c_q]$. Fixed literal retention at the same budget has error $(1-\rho)c_q + O(m^{-1})$. For $q = 2$ and $\rho = 1/2$, the limiting errors are $1/4$ and $h_2^{-1}(1/2) \approx 0.110028$, respectively, with h_2^{-1} taken on $[0, 1/2]$. For fixed $\rho \in (0,1)$, the subsequent limit $q \to \infty$ gives

$$(1-\rho)(1-1/q) - D_q(\rho) = \frac{h_2(1-\rho)}{\log_2 q} + o\left(\frac{1}{\log q}\right). \tag{25}$$

The proof is in Appendix B.8.

For fixed literal retention, the set R is chosen in advance and is public. The cache stores exactly X_R, with no information about other coordinates.

Corollary B.10 (Fixed literal retention frontier)**.** *For uniform independent q-ary records and $I \sim \pi$ independent of the database, a fixed retained set R achieves*

$$\mathrm{Acc}_\pi(R) = \frac{1}{q} + \left(1 - \frac{1}{q}\right) \pi(R). \tag{26}$$

Here $\pi(R) = \sum_{i \in R} \pi_i$. For an integer $0 \le s \le m$, among all sets with $|R| \le s$, the optimum retains the s largest query probabilities. For binary records, $\mathrm{Acc}_\pi^*(s) = \frac{1}{2} + \frac{1}{2} \sum_{j=1}^{s} \pi_{(j)}$, where $\pi_{(1)} \ge \cdots \ge \pi_{(m)}$. Thus, uniform queries give $\mathrm{Acc}_{\mathrm{unif}}^*(s) = \frac{1}{2} + \frac{s}{2m}$, while a Zipf law $\pi_i = i^{-\alpha}/H_{m,\alpha}$ gives $\mathrm{Acc}_{\mathrm{Zipf}}^*(s) = \frac{1}{2} + \frac{H_{s,\alpha}}{2H_{m,\alpha}}$. We use $\alpha \ge 0$ and $H_{s,\alpha} = \sum_{j=1}^{s} j^{-\alpha}$, with $H_{0,\alpha} = 0$. The full proof can be found in Appendix B.13.

Proof of Corollary B.9. On $[0, c_q]$, f_q is continuous and strictly decreasing from $\log_2 q$ to zero. The converse implies that the minimum uniform-query error at budget B_m is at least $f_q^{-1}(B_m/m)$, which converges to $D_q(\rho)$. For any fixed $D > D_q(\rho)$ with $D < c_q$, choose $\gamma > 0$ so that $f_q(D) + \gamma < \rho \log_2 q$. By part (c) of Theorem 2.2, a code of error at most D fits in B_m for all sufficiently large m. Letting $D \downarrow D_q(\rho)$ proves convergence. Literal retention stores $s_m = \lfloor B_m / \log_2 q \rfloor$ coordinates and incurs error $(1 - s_m/m)c_q$ by Corollary B.10, yielding the asserted $O(m^{-1})$ term.

For the alphabet limit, set $a = 1 - \rho$. The defining equation for $D_q = D_q(\rho)$ gives the exact identity

$$a - D_q = \frac{h_2(D_q) + D_q \log_2(1 - 1/q)}{\log_2 q}. \tag{27}$$

Since $0 \le h_2(D_q) \le 1$ and $|\log_2(1 - 1/q)| = O(q^{-1})$, this implies $D_q \to a$. Continuity then gives $h_2(D_q) = h_2(a) + o(1)$. Subtracting a/q from (27) proves (25). Here $h_2(a) > 0$ for fixed $\rho \in (0, 1)$, so the leading gap has order $1/\log_2 q$. □

B.9 Perfect-code summaries at finite length

Proof of Proposition 2.4. Let $\mathcal{K} \subseteq \{0,1\}^n$ be a binary perfect code with 2^k codewords and covering radius t, so that its radius-t balls partition $\{0,1\}^n$. Here $n \ge 1$, $0 \le k \le n$, and $0 \le t \le n$ are integers. Linearity is not required. The expected Hamming loss of a uniform block is

$$\mu(n, t) = \frac{\sum_{d=0}^{t} d\binom{n}{d}}{\sum_{d=0}^{t} \binom{n}{d}}. \tag{28}$$

For fixed disjoint blocks, take b_j copies of perfect codes of length n_j, cardinality 2^{k_j}, and radius t_j, together with a_1 literal bits and a_0 guessed bits, where all counts are nonnegative integers and $a_0 + a_1 + \sum_j b_j n_j = m$. The codebooks and block assignment are public and independent of X. The composed deterministic cache attains

$$S = a_1 + \sum_j b_j k_j, \qquad \bar{e} = \frac{a_0/2 + \sum_j b_j \mu(n_j, t_j)}{m}. \tag{29}$$

For a perfect code, the radius-t balls are disjoint and cover the binary cube. Thus, each source block has a unique nearest codeword. Every ball contains $V(n, t) = \sum_{d=0}^{t} \binom{n}{d}$ words, of which $\binom{n}{d}$ have distance d from the center. For a uniform source block, the probability of distance d is therefore $\binom{n}{d}/V(n, t)$, proving (28). Every center has a nonempty preimage, so the block index has exactly 2^k possible values. Independent disjoint source blocks give a product support. Literal bits each contribute two states, and a fixed guess contributes one. Consequently the composed cache has exactly $2^{a_1 + \sum_j b_j k_j}$ states. Additivity of expected Hamming loss and uniform querying prove (29). No block selector or random seed survives beyond the public fixed assignment.

For Hamming codes, $n = 2^r - 1$, $k = n - r$, and $t = 1$, so $\mu = n/(n+1)$ and $\mu/n = 2^{-r}$. For the binary Golay code, $n = 23$, $k = 12$, and $t = 3$, giving

$$V(23, 3) = 1 + 23 + 253 + 1771 = 2048, \qquad \frac{\mu(23, 3)}{23} = \frac{23 + 2 \cdot 253 + 3 \cdot 1771}{23 \cdot 2048} = \frac{127}{1024}. \tag{30}$$

For R_{2t+1}, every word has weight at most t or distance at most t from the all-one word, and these alternatives are disjoint. Thus, the two radius-t balls partition the cube, and each contains 2^{2t} words. The identity

$$\sum_{d=0}^{t} d\binom{2t+1}{d} = (2t+1)\sum_{u=0}^{t-1} \binom{2t}{u} = \frac{2t+1}{2}\left(2^{2t} - \binom{2t}{t}\right) \tag{31}$$

gives $\rho = 1/(2t+1)$, $\bar{e} = 1/2 - \binom{2t}{t}/(2^{2t+1})$ after division by $(2t+1)2^{2t}$. Literal and guessed bits cover the trivial full-space and singleton codes.

At $m = 128$ and $S = 64$, five Golay indices and four R_3 indices use $5 \cdot 12 + 4 \cdot 1 = 64$ bits. They cover $5 \cdot 23 + 4 \cdot 3 = 127$ records. Guessing the remaining bit gives expected loss $5(2921/1024) + 4(3/4) + 1/2 = 18189/1024$ and accuracy $1 - 18189/131072 = 112883/131072$.

For the high-rate statement, set $a_r = 2^{-r}$ and $d_r = h_2^{-1}(r/(2^r - 1))$. As $r \to \infty$, $h_2(a_r) \sim r2^{-r}$ and $r/(2^r - 1) \sim r2^{-r}$. For every fixed $\epsilon \in (0, 1)$,

$$\frac{h_2((1 \pm \epsilon)a_r)}{r2^{-r}} \longrightarrow 1 \pm \epsilon, \tag{32}$$

using $h_2(x) = x \log_2(1/x) + O(x)$ as $x \downarrow 0$. Monotonicity of h_2 on $[0, 1/2]$ then puts d_r between $(1 - \epsilon)a_r$ and $(1 + \epsilon)a_r$ for all sufficiently large r. Thus, $d_r/a_r \to 1$. This is a high-rate limit, with $\rho \to 1$, and does not establish finite-length optimality at a fixed budget fraction. □

B.10 ACCOUNTING FOR VALUE-DEPENDENT SELECTION

Proof of Proposition 2.3. There are $\binom{m}{k}$ possible selected sets and at most q^k value tuples per set. An injective binary encoding of these possibilities gives B_{sel}. Applying the uniform-query converse to this representation gives $\mathrm{Acc} \le 1 - f_q^{-1}(\min\{B_{\mathrm{sel}}/m, \log_2 q\})$ for a MAP decoder. Any other decoder has no greater accuracy.

Under the product law $P_X P_R$, condition on $R = r$. Each coordinate outside r is uniform and independent of X_r, so the optimal accuracy is $1/q + c_q k/m$. For a decoder ϕ, average over the uniform query and fresh decoding coins to define its accuracy score $g_\phi(x, r) \in [0, 1]$. Total variation gives

$$\mathbb{E}_{P_{X,R}} g_\phi \le \mathbb{E}_{P_X P_R} g_\phi + \epsilon_{\mathrm{sel}} \le A_{\mathrm{lit}} + \epsilon_{\mathrm{sel}}. \tag{33}$$

Maximizing over decoders proves the claimed upper bound on $\mathrm{Acc}_{\mathrm{MAP}} - A_{\mathrm{lit}}$. Under any selection law, decoding retained coordinates exactly and independently guessing a uniform symbol on every miss attains A_{lit}. This proves the lower bound. The cases $k = 0$ and $k = m$ follow from the same argument. □

Example B.11 (Information in selected indices)**.** With two independent uniform bits and $k = 1$, select $R = \{1\}$ when $X_1 = X_2$ and $R = \{2\}$ otherwise. The pair (R, X_R) distinguishes all four databases, so both coordinates can be recovered exactly. The selected value alone is one bit, while the complete state uses two. This shows why a selection pattern cannot be omitted when it depends on the database.

B.11 TIMING FOR REPEATED REQUESTS

Corollary B.12 (Mean error across repeated requests)**.** *Let $m \ge 1$, $q \ge 2$, and $S \ge 0$ be integers. Let X be uniform on $[q]^m$, and let a cache C with at most 2^S states be generated from X and independent encoder randomness before queries $I_1, \dots, I_r$ arrive. Assume $r \ge 1$ is an integer, the queries are independent draws from π and are independent of (X, C), and the response procedure uses only C, the queries, and fresh randomness. It receives no further information about X. If its mean population error per request is at most $\bar{e} \in [0, c_q]$, then*

$$S \ge \lceil \mathcal{R}_\pi(\bar{e}) \rceil . \tag{34}$$

Conversely, any cache and coordinate decoders achieving error $\bar{e}$ under π can be reused for all r requests with the same mean error and the same initial cache size.

For a fixed public set $J \subseteq [m]$ of $r \le m$ distinct indices known when the cache is built, exact recovery of all answers $(X_i)_{i \in J}$ requires and suffices with $\lceil r \log_2 q \rceil$ boundary bits. Therefore

$$\lceil r \log_2 q \rceil < \lceil \mathcal{R}_\pi(\bar{e}) \rceil \tag{35}$$

is a sufficient condition for this known-query cache to use fewer bits than every pre-query cache meeting the mean-error target.

Proof. Draw an independent uniform request position $T \in [r]$. To decode a particular coordinate i from C, set $I_T = i$, draw all other queries independently from π, and simulate the response procedure up to request T, using fresh coins. These auxiliary variables are independent of (X, C), and the procedure receives no new source information. Its coordinate errors e_i satisfy

$$\sum_i \pi_i e_i = \frac{1}{r} \sum_{t=1}^{r} \Pr[\widehat{X}_{I_t} \ne X_{I_t}] \le \bar{e}. \tag{36}$$

Applying Theorem 2.2 and then taking a single ceiling proves (34). Reusing the original coordinate decoder with fresh coins proves the achievability statement for mean error. No independence between response errors is needed.

For fixed J, the q^r distinct answer tuples must have disjoint cache supports under exact decoding. Hence, at least q^r states are needed, and storing the tuple attains this number. Comparing the two integer budgets proves (35). The statement concerns the mean error per request, not the probability that any of the r answers is wrong. An answer-only cache for J leaves every outside coordinate independent and uniform, so it supplies no better-than-chance recall for an independent query outside J. A later request can instead use additional retained information or a newly accessed source, whose persistent representation must also be counted. □

B.12 Full proof of Corollary B.13 and Remark B.14

Here, we illustrate the main worst-case consequence of Theorem 2.2. Let $\ell_m = \max\{1, \lceil \log_2 m \rceil\}$, and let $\mathrm{bin}_{\ell_m}(i)$ be the fixed-width binary encoding of $i-1$ for $i \in [m] = \{1, \dots, m\}$. Define the valid associative-recall language

$$\mathrm{AR}_m = \{x\#\,\mathrm{bin}_{\ell_m}(i)\#b : x \in \{0,1\}^m,\ i \in [m],\ b = x_i\}\,.$$

Malformed codes are rejected. Every valid input has length $N(m) = m + \ell_m + 3$.

Corollary B.13 (Worst-case associative recall)**.** *If a private-state randomized decoder recognizes AR_m with worst-case error at most $\varepsilon \in [0, 1/2)$, then its cache budget $S = S_D(N(m), m)$ immediately after x satisfies $S \geq m\big(1 - h_2(\varepsilon)\big)$. Storing all m bits of x gives a matching $O(m)$ upper bound, meaning that for every fixed $0 \leq \varepsilon < 1/2$, $K^{\mathrm{priv},\varepsilon}_{\mathrm{AR}_m}(N(m)) = \Theta(m)$ as $m \to \infty$. The full proof can be found in Appendix B.12.*

Remark B.14 (KV-bit accounting). Suppose that, at the database split, an implementation stores at most k full KV positions for each of L layers and h heads, with head dimension d_h, b bits per scalar, and at most M additional persistent metadata bits. Here L, h, d_h, b are positive integers, k, M are nonnegative integers, and M includes all surviving information outside the tensors, including occupancy, indices, summaries, and reusable randomness. The total persistent representation then contains at most $2Lhd_hbk + M$ bits. If this state solves AR_m with worst-case error at most $\varepsilon \in [0, 1/2)$, then

$$2Lhd_hbk + M \geq m\big(1 - h_2(\varepsilon)\big), \tag{37}$$

leading to

$$k \geq \frac{\big[m(1 - h_2(\varepsilon)) - M\big]_+}{2Lhd_hb}. \tag{38}$$

The full proof can be found in Appendix B.12.

Proof. For Corollary B.13, let X be uniform on $\{0,1\}^m$, and run the decoder through the database prefix. To recover X_i from its state, append the fixed suffix $\#\,\mathrm{bin}_{\ell_m}(i)\#1$ and predict 1 exactly when the language recognizer accepts. If $X_i = 1$, the complete string is a member of AR_m. If $X_i = 0$, it is not. The worst-case recognition guarantee therefore gives coordinate error $e_i \leq \varepsilon$ for every i. Theorem 2.2 yields

$$S \geq \sum_{i=1}^{m} (1 - h_2(e_i)) \geq m(1 - h_2(\varepsilon)),$$

because h_2 is increasing on $[0, 1/2]$. The upper bound stores all m database bits, parses the fixed-width query, and compares the requested bit. A validity flag rejects malformed blocks, delimiters, and out-of-range indices. Its maximum state is $m + O(\log(m+1)) = O(m)$ bits.

For Remark B.14, each full KV position contains one key and one value vector for each layer and head. The number of stored scalar values is $2Lhd_hk$, so at b bits per scalar the tensors have at most $2Lhd_hbk$ bits. Adding M metadata bits leaves at most

$$2^{2Lhd_hbk+M}$$

distinct persistent states. Applying Theorem 2.2 with $S = 2Lhd_hbk + M$ to the coordinate decoders just constructed gives (37). Dividing by the positive scalar $2Lhd_hb$ and using $k \geq 0$ gives (38). Here $[v]_+ = \max\{v, 0\}$. An integer slot budget may be rounded upward after this division. □

B.13 Full proof of Corollary B.10

Proof. Condition on a query $I = i$ with $\pi_i > 0$. If $i \in R$, the cache contains X_i and the answer is correct with probability one. If $i \notin R$, then X_i is independent of X_R and uniform on $[q]$, even after conditioning on $I = i$. No function of the retained records can predict it with probability greater

than $1/q$. Therefore

$$\begin{aligned}\mathrm{Acc}_\pi(R) &= \sum_{i\in R}\pi_i + \frac{1}{q}\sum_{i\notin R}\pi_i \\ &= \pi(R) + \frac{1}{q}(1-\pi(R)) \\ &= \frac{1}{q} + \left(1-\frac{1}{q}\right)\pi(R).\end{aligned}$$

Adding a coordinate cannot decrease accuracy, so a maximizing set can be taken to have size s. Suppose $|R| = s$ and there are $i \in R$, $j \notin R$ with $\pi_j > \pi_i$. Replacing i by j strictly increases $\pi(R)$. Repeating this exchange leaves the s largest probabilities. The binary, uniform, and Zipf formulas follow by substituting $q = 2$, $\pi_i = 1/m$, and $\pi_i = i^{-\alpha}/H_{m,\alpha}$, respectively. Ties may be broken arbitrarily, and the formulas include $s = 0$ and $s = m$. □

B.14 A LENGTH-INDEXED HIERARCHY

Remark B.15 (Padded INDEX scales). For any positive integer-valued f satisfying $\lceil \log_2 n \rceil \le f(n) \le n$ at all sufficiently large lengths, set $m = \max\{1, \lfloor f(n)/4 \rfloor\}$, $\ell = \max\{1, \lceil \log_2 m \rceil\}$, and $p = n - m - \ell - 3$, and let L_f contain exactly the strings $x0^p\#\,\mathrm{bin}_\ell(i)\#b$ with $x \in \{0,1\}^m$, $i \in [m]$, and $b = x_i$, taking the finitely many remaining length slices to be empty. For every fixed $\varepsilon \in [0, 1/2)$, storing x and checking the suffix uses $O(f(n))$ bits, while the padded INDEX reduction and Corollary B.13 require $\Omega_\varepsilon(f(n))$ bits after x, giving

$$K_{L_f}^{\mathrm{priv},\varepsilon}(n) = \Theta_\varepsilon(f(n)) \tag{39}$$

and separating this scale from every $g = o(f)$ in the same length-nonuniform model.

C DETAILED EXPERIMENTAL SETTINGS AND RESULTS

C.1 EXPERIMENT 1: PERSISTENT MEMORY ACROSS FORMAL LANGUAGES

Shared memory accounting. Each recognizer processes the input once, from left to right, and predicts membership from its final state. The budget counts all logical bits retained between symbols, including private random choices that are needed again. Fixed network weights, fixed precision assignments, and temporary arithmetic within a transition are excluded. The regular-language controllers and Dyck-(2) GRUs receive only the current symbol and their previous state. Equality controllers additionally use the public position to identify the block and its index, and the MMN baseline uses the declared input length to set its capacities. Public random masks are available without a memory charge only in the explicitly public-coin equality baseline.

Test benchmarks, training data, and validation data. We use the term test benchmark for the generated inputs on which recognition performance is measured. Training data supply the examples used to fit model parameters, while validation data select checkpoints and precision allocations. We use seeds 47, 202, and 708. Each seed determines a separate data realization and the corresponding model initialization or randomized evaluation stream. A model is evaluated on its matching benchmark seed, and benchmark inputs are reused across budgets whenever the test distribution is budget independent. Thus, variation across seeds includes data variation as well as training or fingerprint randomness. Training and validation pools are generated before optimization and remain fixed during training. Their sampling and overlap rules are specified for each task below.

C.1.1 TOMITA7 AND MOD31 SETTINGS

Test benchmarks and memory budgets. The Tomita7 test benchmark evaluates recognition of $0^*1^*0^*1^*$, while the Mod31 test benchmark evaluates whether a binary string's numerical value is divisible by 31, allowing leading zeros. Both benchmarks use lengths $n \in \{16, 32, 64, 128, 256\}$. For every length and seed, each benchmark contains 256 positive and 256 negative test strings, sampled without replacement within that set. The evaluation budget grids are $B \in \{1, 2, 3, 4, 5\}$ for

Tomita7 and $B \in \{3, 4, 5, 6, 7\}$ for Mod31. One controller is trained per task, budget, and seed and evaluated at all five benchmark lengths.

Positive Tomita7 test strings cover the valid run patterns 0, 1, 01, 10, 010, 101, and 0101. Their allocation across one through four runs uses weights $(0.1, 0.2, 0.3, 0.4)$, adjusted for the number of distinct strings available. Run boundaries are sampled uniformly. Negative test strings are stratified by the first position at which the reference DFA enters its invalid state. The early, middle, late, and final-position strata contain 26, 51, 77, and 102 examples, respectively, with the first two boundaries at $\lfloor n/3 \rfloor$ and $\lfloor 2n/3 \rfloor$. Each negative starts with a valid prefix $0^a1^b0^c1^d$, where $a \geq 0$ and $b, c, d \geq 1$, appends an invalid zero, and then receives a random tail. In the Mod31 test benchmark, all positives have remainder zero, and the negatives are distributed as evenly as possible over remainders 1 through 30, giving eight or nine examples per nonzero remainder. Sampling within each remainder class is uniform. Test labels are checked independently against the language definitions.

Neural controller architecture. The deployed controller retains B signs in $\{-1, +1\}^B$. Its transition network takes these signs and a two-dimensional one-hot symbol, applies two 128-unit $\tanh$ layers, and produces B outputs that are thresholded to signs. A separate $B \to 128 \to 1$ network with a $\tanh$ hidden layer predicts acceptance using a zero logit threshold. The initial cache is learned, and no DFA state or position counter is supplied to the controller.

Training and validation data. The encoder and transition stages use a sampled prefix–suffix training curriculum. An observation vector is formed for each training prefix by querying membership after appending each sampled suffix. Each seed uses 512 prefixes, including the empty prefix, and 256 suffixes. The one-symbol extensions of every prefix are queried as well. For Tomita7, training prefix and suffix lengths are sampled from zero through seven, with an explicit empty suffix. For Mod31, nonempty training prefixes have length 17 and suffixes have length 13. A $256 \to 128 \to B$ encoder and a $B \to 128 \to 256$ decoder learn to reconstruct these observation vectors. Repeated observation vectors are deduplicated, but their binary encodings are learned. This training curriculum changes with the seed and is shared across budgets within a seed. All semantic supervision consists of membership labels, without DFA state identifiers or an explicit modular transition rule.

The final acceptance-head stage uses additional balanced pools of 512 training and 128 validation strings at each of the five benchmark lengths, totaling 2,560 and 640 strings per seed. To generate these training and validation pools, Tomita7 proposals use up to eight runs before membership-based rejection sampling, while Mod31 uses uniform binary proposals. The two pools are mutually disjoint and generated independently of the test benchmarks, with accidental overlap with benchmark inputs permitted. Training strings fit the acceptance head, and validation strings select its checkpoint.

Training and checkpoint selection. The encoder is trained for 1,500 full-batch epochs with Adam at learning rate 0.003. Each epoch processes the entire fixed set of distinct observation vectors in one batch, so it corresponds to exactly one optimizer update. Its loss combines binary cross-entropy reconstruction, a collision penalty with weight 10, and a penalty encouraging squared state values to approach one. Positive reconstruction labels have weight 20, and the binary penalty has maximum weight 0.01, multiplied at update t by $\min(t/2000, 1)$. For two distinct observation vectors with continuous encodings u and v, the soft collision score is $\prod_{k=1}^{B} \frac{1+u_k v_k}{2}$. Tomita7 penalizes this score directly. Mod31 uses its negative-log complement, $-\log(1 - p)$ for collision score p, to increase the penalty near a collision. In both cases, penalties are summed over ordered pairs of distinct observation vectors and divided by the number of distinct vectors.

Next, the transition and acceptance networks are trained for 1,500 full-batch epochs with Adam at learning rate 0.003, using the learned parent and child encodings as transition targets and prefix membership as acceptance targets. Each epoch processes all sampled parent–child transitions and prefix membership targets in one batch, again giving one optimizer update per epoch. The first 750 epochs use continuous states and the remaining 750 use hard forward states with straight-through gradients. Transition training clips the gradient norm at 5. Finally, transitions are frozen and the acceptance head is trained on the complete-string training pool for 20 epochs with Adam at learning rate 0.001. Each head epoch visits every training example once in five length-specific batches, giving 3,100 optimizer updates across all stages. The head checkpoint with highest validation accuracy is selected, breaking ties by lower validation cross-entropy.

Table 2: Test accuracy on Tomita7 in Experiment 1. Entries show positive accuracy / negative accuracy, computed within each true class. Each value is the mean ± sample standard deviation across seeds 47, 202, and 708. Each length and seed uses 256 positive and 256 negative test strings. Zero standard deviations are omitted. A single 100.0 means both classes have perfect accuracy for every seed. All accuracies are percentages.

Length n	Positive / negative accuracy (%)				
	$B = 1$	$B = 2$	$B = 3$	$B = 4$	$B = 5$
16	100.0 / 0.0	28.4 ± 1.8 / 100.0	100.0	100.0	100.0
32	100.0 / 0.0	30.5 ± 1.8 / 100.0	100.0	100.0	100.0
64	100.0 / 0.0	35.2 ± 1.7 / 100.0	100.0	100.0	100.0
128	100.0 / 0.0	38.0 ± 1.6 / 100.0	100.0	100.0	100.0
256	100.0 / 0.0	37.8 ± 2.5 / 100.0	100.0	100.0	100.0

Table 3: Test accuracy on Mod31 in Experiment 1. Entries show positive accuracy / negative accuracy, computed within each true class. Each value is the mean ± sample standard deviation across seeds 47, 202, and 708. Each length and seed uses 256 positive and 256 negative test strings. Zero standard deviations are omitted. A single 100.0 means both classes have perfect accuracy for every seed. All accuracies are percentages.

Length n	Positive / negative accuracy (%)				
	$B = 3$	$B = 4$	$B = 5$	$B = 6$	$B = 7$
16	33.9 ± 36.6 / 66.3 ± 36.4	2.7 ± 4.7 / 97.4 ± 4.5	100.0	100.0	100.0
32	36.5 ± 42.8 / 64.3 ± 41.0	2.2 ± 3.8 / 97.0 ± 5.2	100.0	100.0	100.0
64	36.2 ± 39.9 / 62.6 ± 41.3	3.1 ± 5.4 / 97.5 ± 4.3	100.0	100.0	100.0
128	36.5 ± 40.5 / 64.3 ± 37.2	2.9 ± 5.0 / 96.7 ± 5.6	100.0	100.0	100.0
256	37.5 ± 40.3 / 63.9 ± 40.6	2.6 ± 4.5 / 97.4 ± 4.5	100.0	100.0	100.0

C.1.2 TOMITA7 AND MOD31 RESULTS

Tables 2 and 3 show that the continuation-bottleneck controllers achieve 100% accuracy on both positive and negative test examples at every tested budget $B \geq 3$ for Tomita7 and $B \geq 5$ for Mod31. This holds at all five input lengths from 16 to 256 and for all three seeds. Below these budgets, the errors are strongly class dependent. The one-bit Tomita7 controller accepts every test string, while the two-bit controller correctly rejects every negative but recognizes only 28.4%–38.0% of positives. For Mod31, the three-bit controller attains 33.9%–37.5% positive accuracy and 62.6%–66.3% negative accuracy. At four bits, these ranges become 2.2%–3.1% and 96.7%–97.5%, respectively. These ranges are across input lengths after averaging each class's accuracy over the three seeds. The large class-specific standard deviations for Mod31 at three bits further show that similar overall accuracies can conceal different errors across seeds.

Corollary B.2 explains the exact memory requirements through future distinguishability. Two prefixes of the same length are equivalent when every suffix of the required remaining length gives the same membership decision. Tomita7, whose language is $0^*1^*0^*1^*$, has four valid stages corresponding to the initial zero block, the first one block, the second zero block, and the final one block, together with an absorbing invalid state. For Mod31, a prefix's future behavior is determined by its remainder modulo 31, and different remainders can be distinguished by a common suffix. At every evaluated input length, all five Tomita7 classes and all 31 Mod31 classes are reachable after five symbols and remain distinguishable by suffixes of the remaining length. Thus, the theorem gives exact requirements of $\lceil \log_2 5 \rceil = 3$ and $\lceil \log_2 31 \rceil = 5$ bits, with matching DFA constructions supplied by Proposition B.3. By part (a) of Theorem 2.1, the learned caches at the threshold budgets therefore give optimal deterministic one-way messages at that split, since the recipient can resume recognition on any remaining suffix from the transmitted state. A smaller cache must merge prefixes that require different answers for some shared suffix, making exact recognition impossible. The particular positive and negative error rates, however, depend on the learned representation and the test distribution.

For every tested budget at or above these thresholds, an exhaustive traversal of the reachable controller–oracle state pairs confirms agreement on all binary strings, with exactly 5 reachable con-

troller states for Tomita7 and 31 for Mod31. Every cache bit varies even at larger tested budgets, so the unchanged state counts reveal redundancy across bits. Under the state-cardinality definition of cache size, the reachable codes still admit 3- and 5-bit identifiers, although the implementation retains all B signs. The controllers learn their state encodings and transitions from membership labels for sampled prefix–suffix queries and complete strings, without receiving DFA state identifiers, transition tables, or an explicit modular update rule. Recovering minimal exact state spaces from this supervision suggests that the same behavior may extend to other regular-language recognition problems. An exact solution may be likewise discovered once its memory budget is sufficient to represent all necessary future-equivalence classes.

C.1.3 EQUALITY SETTINGS

Mixed and structured stress test benchmarks. We evaluate equality on two generated test benchmarks, one with mixed negatives and one with structured stress negatives. Both use inputs of the promised form $x\#y$, with $x, y \in \{0,1\}^m$, $m \in \{8, 16, 32, 64, 128, 256\}$, and total length $n = 2m + 1$. Each block length, test distribution, and seed uses $4{,}096$ benchmark examples, split into eight evaluation batches containing 256 positives and 256 negatives each. Positive test examples draw x uniformly and set $y = x$. Mixed test negatives choose with equal probability among a random unequal block, a swap of adjacent opposite bits, a swap of two randomly positioned opposite bits, a swap between opposite endpoint bits, and bitwise complementation. Benchmark sampling is with replacement, which is necessary to retain the stated sample count at small m.

The structured stress test benchmark depends on the polynomial budget. For $B \in \{4, 6, 8, 10, 12, 14\}$, let q be the largest prime below $2^{B/2}$, giving $q \in \{3, 7, 13, 31, 61, 127\}$. Choose the largest positive divisor s of $q - 1$ satisfying $s < m$, and create a negative test example by swapping two opposite bits at positions separated by s. The first position and orientation of the opposite bits are uniform, and the remaining bits are uniform. Each budget has its own stress test negatives, generated independently of the realized fingerprints. Within a seed and length, mixed benchmark examples are reused across budgets and positive test examples are shared across the mixed and stress benchmarks. This pairing supports direct comparisons without treating reused benchmark inputs as additional independent samples.

Baseline algorithms and test evaluation. The exact equality oracle stores x in m bits and reuses a consumed buffer bit as a mismatch flag while reading y. The private polynomial baseline instead stores only $(\alpha, d) \in \mathbb{F}_q^2$, where α is drawn uniformly from the nonzero field elements and d starts at zero. At public position $i \in \{0, \ldots, m-1\}$, it adds $x_i\alpha^i$ while reading x and subtracts $y_i\alpha^i$ while reading y, modulo q. The separator leaves the state unchanged, and the controller accepts when the final d is zero. Powers are recomputed from the stored α and public position, so the persistent budget is exactly $2\lceil \log_2 q \rceil = B$ bits.

The public-coin baseline uses budgets $B \in \{1, 2, \ldots, 10\}$ and an external uniform binary mask of shape $B \times m$. It maintains B parity checks, using the same mask column for corresponding positions of x and y, and accepts when all checks are zero. Both fingerprints always accept equal blocks. The public baseline has false-acceptance probability 2^{-B} on each fixed unequal pair, whereas the polynomial collision probability depends on the difference polynomial and field size. Each randomized baseline uses eight fresh fingerprint trials per example and matching seed. Thus, a condition contains $32{,}768$ trial outcomes per seed, while still containing only $4{,}096$ sampled inputs.

Neural controller architecture. The neural comparison uses the same even budgets as the polynomial baseline. Its persistent state consists of two $b = B/2$-bit codes, each with $S = 2^b$ possible values and initialized to zero. A shared recurrent cell updates the first code while reading x and the second while reading y. The current code and binary symbol index a learned $2S \times 64$ embedding with a $\tanh$ activation, followed by a 64-unit $\tanh$ layer and an S-way output. The final head takes the absolute difference of the two binary code vectors, applies two 64-unit $\tanh$ layers, and outputs an acceptance logit.

Training and validation data. Every neural model trains jointly across all six benchmark block lengths using separately generated pools of $2{,}048$ training and $1{,}024$ validation examples per length. These pools are class balanced, use sampling with replacement, and are shared across budgets within a seed. The training and validation negative mixture uses the same five transformations as the mixed test benchmark, each selected with probability $1/5$, without a dedicated single-bit-flip component.

These pools are generated independently of each other and of the test benchmarks, permitting accidental overlap. Training examples fit the controller, while validation examples select its checkpoint.

Training and checkpoint selection. Training uses hard categorical transitions with straight-through gradients. Transition scores undergo 30 Sinkhorn row and column normalization iterations at temperature 0.02, followed by rowwise argmax, without projection onto a bijection. The loss is membership binary cross-entropy plus state-balance and symbol-diversity penalties, each with weight one.

Adam uses learning rate 0.0003, batch size 128, and gradient clipping at 5. The straight-through relaxation temperature decreases linearly from 1 to 0.3. Training completes 30 epochs, each visiting all $12{,}288$ training examples once in shuffled order, for 96 updates per epoch and $2{,}880$ updates overall. Validation is performed after each epoch, selecting mean accuracy first, then the lowest stratum accuracy across lengths, then lower cross-entropy. At deployment, normalization is folded into fixed network biases, and only the two hard codes persist.

C.1.4 Equality results

Figure 3 plots polynomial fingerprint error against block length m, the same results against persistent-memory budget B, and public-coin fingerprint error against B.

On mixed negatives, polynomial error generally decreases as the budget increases. This is consistent with the private-state guarantee in Proposition B.3, since a larger budget permits a larger prime modulus q and can reduce fingerprint collisions. However, the error varies nonmonotonically with length, and individual budget increases can raise error. These fluctuations depend on how the swap gaps in the mixture interact with the chosen modulus. For example, at $m = 64$, endpoint swaps collide for half the nonzero field values when $q = 127$ ($B = 14$), helping explain the higher error than at $B = 12$. A larger field can therefore perform worse on a particular negative type. This behavior does not contradict the proposition, which provides a sufficient memory bound for a worst-case error guarantee without requiring monotonic improvement in average benchmark error across these field choices.

We therefore test the polynomial baseline on structured stress negatives. The solid curves rise in steps with length and fall with budget, with a 50% plateau. This stress test is based on choosing, for each field, the largest divisor s of $q - 1$ below m as the swap gap, maximizing collision probability $s/(q-1)$ within the opposite-bit swap family. Longer blocks allow larger divisor gaps, while larger budgets enlarge the field and can reduce the collision fraction. This explains the observed steps and the need to scale the field with length in the polynomial construction of Proposition B.3. Once $m \geq q$, all negatives collide. Since the positive half is always accepted correctly, error is capped at 50%, attained when every negative is falsely accepted.

Public-coin error in panel (c) closely follows $2^{-(B+1)}$, approximately halving per bit with little length dependence. Its advantage at matched budgets (Figure 1a) illustrates the seed accounting in Proposition 2.5, because free mask reuse leaves all B bits for parity checks while private polynomial fingerprints also store α. At $m = 256$, $B = 6$, (13) gives every private-state algorithm worst-case error at least $7/15 \approx 46.67\%$, whereas public checks guarantee at most $1/64 \approx 1.56\%$ on promised inputs. Smaller plotted errors are compatible with this separation because they average over the benchmark. The private bound becomes zero at $B \geq 8$ for this length, so it does not establish an unavoidable gap throughout the figure.

Figure 1a shows neural gains over private fingerprints at $B = 4, 6, 8, 10$, reaching 1.33% pooled error at $B = 10$, $m = 256$; Figure 4 extends to $B = 14$ (0.54%). The former pools three seeds, with 95% percentile CIs from $10{,}000$ bootstrap resamples of 24 batches. At $B = 2$, all three neural controllers implement the polynomial baseline's parity comparison. Because swaps and complementation preserve parity at tested even lengths, while random unequal blocks match about half the time, the balanced mixture explains their overlapping errors near 45%, versus 12.5% expected for public checks. The neural gains illustrate learning distinctions useful on the benchmark. However, the fixed second code leaves only $B/2$ varying bits after the first block, so Theorem 2.1 and Corollary B.2 require $B \geq 2m$ for exact recognition with this architecture. Low average error thus remains compatible with collisions on distinguishing continuations.

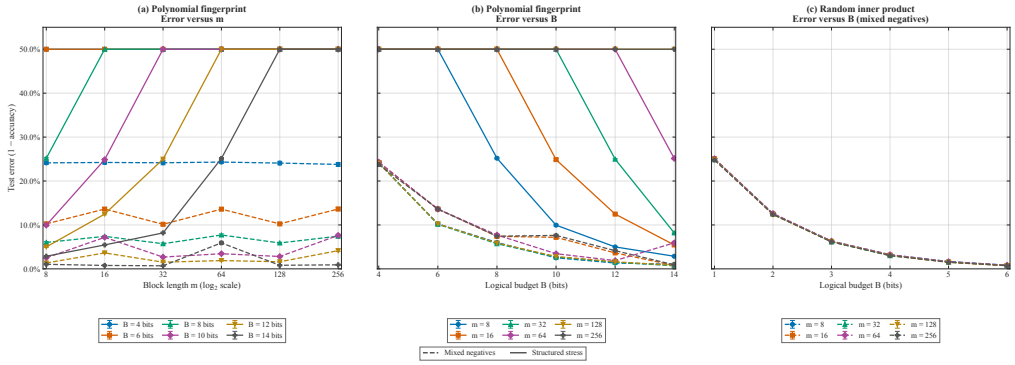


Figure 3: Fingerprint error for the equality task $x\#y$, where x and y are binary blocks of length m. **(a)** Polynomial fingerprint error versus block length, with colors identifying memory budgets B bits. **(b)** Polynomial fingerprint error versus memory budget, with colors identifying block lengths m. **(c)** Public-coin random inner-product fingerprint error versus memory budget B bits on mixed negatives. Dashed curves denote mixed negatives, while solid curves denote structured stress negatives. Curves show error pooled across seeds 47, 202, and 708. Error bars show 95% percentile confidence intervals from 10,000 bootstrap resamples of the 24 pooled evaluation batches, using bootstrap seed 172. Each resample draws whole batches with replacement, retaining all inputs and fingerprint trials within each sampled batch.

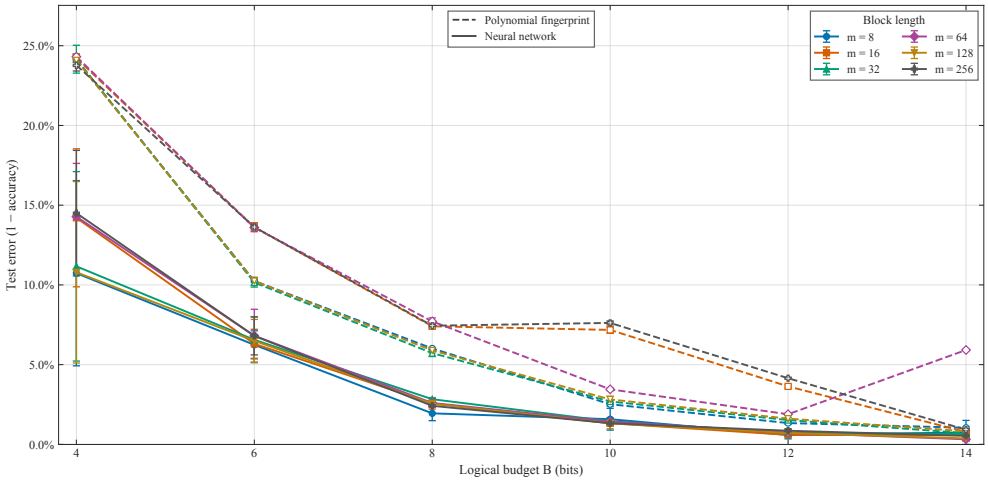


Figure 4: Test error versus logical persistent-memory budget B for the equality task, on the mixed-negative benchmark. Solid curves denote the neural controller, and dashed curves denote the private polynomial fingerprint. Curves show the equal-weight mean error across seeds 47, 202, and 708, and error bars indicate ± 1 sample standard deviation across these seeds.

C.1.5 DYCK-(2) SETTINGS

Eight-family test benchmark. We evaluate the GRU and MMN on a generated eight-family test benchmark at lengths $n \in \{16, 32, 64, 128, 256, 512\}$, using two bracket types encoded as four distinct symbols. Each test family contains matched valid and invalid strings. Six families corrupt a valid source by causing early stack underflow, a bracket-type mismatch, a failure in the final quarter, an unfinished stack, underflow through an endpoint swap that preserves symbol counts, or a swap of two closing brackets of different types. Valid source shapes for these test examples are spread equally across feasible maximum-depth bands with boundaries $\lfloor \sqrt{n}/2 \rfloor$, $\lfloor \sqrt{n} \rfloor$, $\lfloor 2\sqrt{n} \rfloor$, and $n/2$. Test corruptions are retained only when the independent stack oracle confirms the intended failure.

The other two families probe longer-range type matching. Ascension examples contain nested blocks with an internal type query and a compensating valid pair, while distant-swap examples exchange closing brackets belonging to two long sibling subtrees. Their positive and negative members have identical untyped bracket shapes and symbol counts, so these features alone cannot distinguish the labels. These are sampled stress families and do not themselves establish the worst-case lower bound. At $n = 16$, each family contributes 128 positive–negative pairs, or $2{,}048$ test strings per seed across all eight families. At every larger evaluated length, each family contributes $1{,}280$ pairs, giving $20{,}480$ test strings per seed. The GRU and MMN use exactly the same saved test strings.

GRU architecture. The learned recognizer embeds each symbol in 16 dimensions, applies a GRU cell, and predicts membership through a 128-unit $\tanh$ head. Every recurrent coordinate is quantized after every symbol to a finite, uniformly spaced grid in $[-1, 1]$, with straight-through gradients during training. Only this quantized state is retained at inference.

Training and validation data. The GRU uses separate training and validation pools generated from the same eight families as the test benchmark. These pools follow the increasing-length training curriculum

$$16, 20, 24, 28, 32, 40, 48, 56, 64, 80, 96, 112, 128,$$
$$160, 192, 224, 256, 320, 384, 448, 512,$$

up to the desired target length. For each seed and curriculum length, the frozen training pool contains $2{,}048$ examples at length 16, $16{,}384$ at subsequent lengths through 128, and $65{,}536$ above 128. The validation pool contains $2{,}048$ examples at length 16 and $8{,}192$ otherwise. Each pool represents the eight families and both labels equally. Training, validation, and test splits are disjoint within each seed and length. At length 16, limited support requires excluding only the matching seed's benchmark inputs from training and validation. At longer evaluated lengths, these pools exclude every seed's benchmark inputs.

Training and checkpoint selection. A continuous neural GRU initializes each quantized model. Its length-16 bootstrap uses Adam at learning rate 0.003 for at most 200 epochs. Subsequent curriculum stages use AdamW at learning rate 0.001, weight decay 0.01, 64 matched pairs per batch, and at most 100 epochs per stage. The objective is membership cross-entropy plus 0.25 times a matched-pair ranking loss and 0.2 times a next-symbol prediction loss on positive inputs. The ranking term is $\log(1 + \exp(1 - z_+ + z_-))$, where z_+ and z_- are the paired acceptance logits. These targets come from the input strings and labels. At the target length, the quantized student is initialized from this neural teacher and trained for at most 80 epochs at learning rate 0.0005 with $1{,}024$ pairs per batch. The curriculum and student stages clip gradients at 1, halve the learning rate after ten plateau epochs down to 10^{-5}, and select checkpoints by worst-family validation error followed by cross-entropy. After at least 20 epochs, they stop after five consecutive epochs below 1% worst-family validation error or 45 epochs without improvement. The teacher and auxiliary prediction head are used only during training.

GRU precision calibration on training data. The reported models further compress saved GRUs that initially use 16 bits per coordinate. The source widths at lengths $32, 64, 128, 256, 512$ are $10, 18, 31, 29, 31$ coordinates, respectively. Precision calibration uses a frozen, balanced subset of 256 inputs from the existing training pool. At each step, it reduces one coordinate's precision, choosing the reduction that best preserves the source model's soft predictions on this training subset. The explored precision grids are $\{16, 8, 4, 2, 1\}$ and $\{16, 12, 10, 8, 6, 4, 2, 1\}$ bits per coordinate. Network weights remain fixed during this allocation stage, and the validation pool selects the allocation and budget under the criteria below.

GRU refinement on training data. Additional refinement trials continue training a model with a fixed precision allocation for at most 20 epochs using membership and paired-ranking losses. Some selected refinements use Adam at learning rate 0.0001, 128 pairs per batch, and a frozen subset of $4{,}096$ matched pairs from the existing training pool. Other selected refinements use the full training pool at learning rate 0.0003, with 256 pairs per batch below $n = 512$ and $1{,}024$ at $n = 512$. Refinement clips gradients at 1 and uses the validation pool to select the best worst-family checkpoint, including epoch zero. After at least five epochs, it stops following three consecutive epochs below 3% worst-family validation error or ten epochs without improvement. This is continued training from the curriculum checkpoints, so the 20-epoch cap does not represent the full training cost. For a model with coordinate precisions $r_1, \ldots, r_h$, the actual logical cache is $\sum_j r_j$ bits. The reported budget is a common ceiling across seeds rounded up to a multiple of 16. Saved audits replay ev-

ery selected benchmark input from the actual packed bytes and check that packing preserves the recurrent states and predictions exactly.

MMN baseline algorithm and test evaluation. The one-pass algorithm of Magniez et al. (2014) uses block size $L = \lceil \sqrt{n \log_2 n} \rceil$, a temporary stack of L bracket-type bits, and capacity $C = \lfloor n/L \rfloor$ for compressed stack entries. Each compressed entry stores a field fingerprint and a remaining height. Let $F = \lceil \log_2 q \rceil$, $H = \lceil \log_2(n+1) \rceil$, $T = \lceil \log_2(L+1) \rceil$, and $J = \lceil \log_2(C+1) \rceil$. The counted budget is

$$B_{\mathrm{MMN}} = L + 2F + T + C(F+H) + H + J + 1.$$

This includes the private field element, temporary fingerprint, both stacks, depth and occupancy counters, and rejection flag. For a given cap, the implementation chooses the largest permitted prime $q > n$ within its field-width limit. Evaluation starts with the smallest feasible field width and increases it if needed. Each test input and seed uses eight independent trials, with a fresh private field element drawn uniformly from $\mathbb{F}_q$ for each trial.

Validation criteria and budget selection. We assess two GRU budget-selection criteria on the eight-family validation pools. The combined criterion requires error strictly below 5% after pooling the eight equally represented families within every seed. The per-family criterion requires error strictly below 5% for each balanced family in every seed, counting both its valid and invalid members. For each length and seed, we evaluate all distinct saved compression allocations and compatible uniform- or mixed-precision GRU checkpoints on validation without retraining, and select the smallest actual bit count passing each criterion. Ties are broken by pooled or worst-family validation error, respectively, then validation cross-entropy and a fixed candidate identifier. The common ceiling is the maximum selected bit count across seeds rounded up to a multiple of 16; each model retains its selected allocation. We freeze all selections before evaluating each unique selected model once on its matching-seed test benchmark.

Test pooled and per-family errors have 95% percentile confidence intervals from $10{,}000$ whole-batch bootstrap draws with seed 172. For each seed, we split each family's matched pairs into eight equal groups in saved order and combine corresponding groups across families into eight balanced batches. Intervals resample eight batches per seed or all 24 batches for pooled estimates without seed stratification. The intervals condition on the frozen selections. The saved test benchmark was used during candidate development, so these intervals do not describe performance on an untouched confirmation set.

C.1.6 DYCK-(2) RESULTS

Figure 5 compares validation-selected GRU ceilings (solid triangles) with test-selected ceilings (dashed circles) and MMN. Pooled ceilings are unchanged at all six lengths; per-family ceilings change only at $n = 128$ (256 to 224 bits, -32) and $n = 512$ (320 to 336 bits, $+16$). Thus, ten of twelve ceilings remain unchanged, and both validation-selected curves are nondecreasing.

Test-selected budgets are labeled optimistic because evaluation data also guided selection; the label does not predict the direction of budget changes. Moreover, validation selection searches all saved allocations and allows different recipes across seeds, so budget shifts do not isolate the selection split. Each common ceiling covers models with different allocations; consequently, equal ceilings need not imply equal test errors.

Figure 6 shows maximum test error across seeds for pooled-selected models (left) and per-family-selected models (right). On the left, at $n = 512$, the unchanged 272-bit ceiling gives mean pooled test error 3.67% (95% CI $[3.17, 4.24]\%$), versus 2.38% under test selection, while maximum ascension error rises from 14.57% to 21.21%. Pooling gives each family one eighth of the weight, allowing easier families to conceal these failures. Because ascension pairs share untyped shapes and symbol counts, their errors suggest difficulty retaining or using distant bracket-type matches. At $n = 64, 128$, only two seeds pass the pooled test criterion.

On the right, at $n = 512$, the 336-bit ceiling yields mean pooled test error 1.34% (95% CI $[1.16, 1.53]\%$) and maximum ascension error 5.70%. All seeds pass the per-family test criterion only at $n = 256$, and none passes at $n = 128$ despite its smaller ceiling. Thus, stricter validation selection improves difficult-family performance without ensuring the test criterion; the improve-

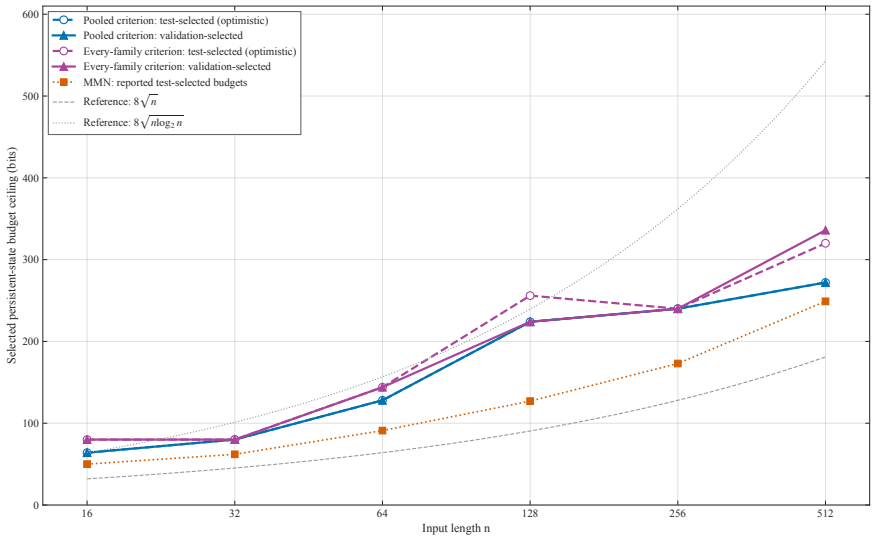


Figure 5: Selected persistent state budget ceilings B versus input length n for the learned GRU and MMN on Dyck-(2). Blue and purple GRU curves use pooled and per-family error strictly below 5% in every seed, respectively. Solid triangles show validation-selected budgets; dashed open circles show test-selected budgets, labeled optimistic because selection used the test benchmark. Validation-selected ceilings are the maximum selected budget across seeds 47, 202, and 708. Orange squares show the reported test-selected MMN budgets. Gray curves give the illustrative bound references.

ment involves separately selected representations and budgets. The intervals condition on frozen selections, and the benchmark was used during development.

The budget growth is qualitatively compatible with Proposition B.3 and (11), which bound worst-case private-state cache complexity between $\Omega(\sqrt{n})$ and $O(\sqrt{n \log n})$ for fixed error and sufficiently large even n. The lower bound constrains every permitted algorithm meeting the worst-case guarantee, including learned transitions. MMN's local stack and cross-block fingerprints implement the upper-bound mechanism; its test-selected curve uses fewer bits at every tested length. However, deterministic GRUs' errors over sampled strings and training seeds do not establish per-input randomized guarantees. Consequently, the observed growth establishes neither the asymptotic rate nor optimal memory. The reference constants are illustrative.

C.2 Experiment 2: Associative recall under a fixed memory budget

This experiment examines how the amount of retained memory and the query distribution affect retrieval accuracy. The task is associative recall: a method first reads a database of indexed values and compresses it into a bounded cache. Only after compression does the method receive an index and attempt to recover the corresponding value from the cache. Because the query is unknown during compression, the retained information must support possible future queries. We compare methods that store selected values exactly with a neural model that learns a compact summary. The comparison tests the fixed-retention prediction in Corollary B.10 and examines learned summaries in relation to the information bound in Theorem 2.2. Varying the query distribution lets us examine how favoring frequently requested records changes accuracy across the full database.

C.2.1 Settings

Test benchmark. The benchmark consists of databases $X = (X_0, \ldots, X_{m-1})$ with m indexed records and queries asking for a value X_i. Each value is drawn independently and uniformly from an alphabet $\{0, \ldots, q-1\}$ of size q. We use database sizes $m \in \{64, 128, 256\}$, alphabet sizes

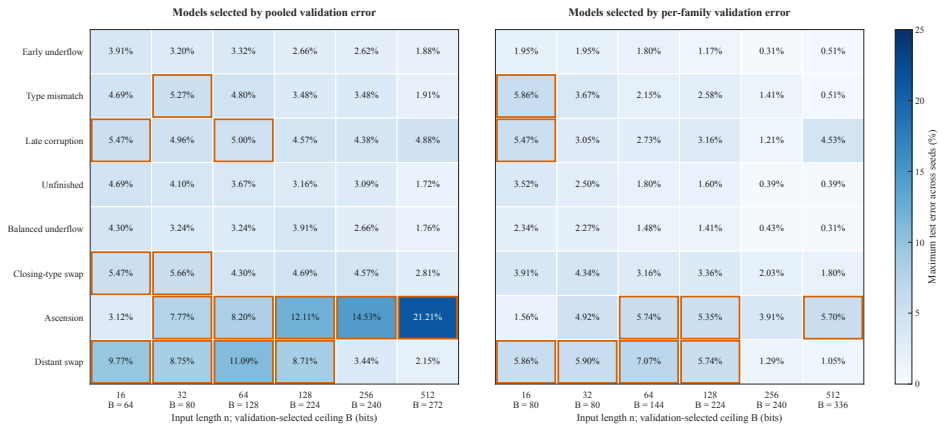


Figure 6: Dyck-(2) test error by input family and length for validation-selected quantized GRUs. **Left.** Models selected for pooled validation error strictly below 5% in every seed. **Right.** Models selected for validation error strictly below 5% in every family and every seed. Each cell shows the maximum family test error across seeds 47, 202, and 708, computed over both valid and invalid examples. Columns give input length n and the validation-selected persistent-memory budget ceiling B in bits. Both panels share a color scale. Orange outlines mark test error at or above 5%.

$q \in \{2, 16, 64\}$, and process seeds 47, 202, and 708. Let π_i denote the target probability of querying coordinate i. The target query law is either uniform, with $\pi_i = 1/m$, or Zipf, with

$$\pi_i = \frac{(i+1)^{-\alpha}}{\sum_{j=0}^{m-1}(j+1)^{-\alpha}}, \qquad \alpha \in \{0.8, 1.1, 1.5\}.$$

For each seed and (m, q) pair, the test benchmark contains 256 independently generated databases. Records are serialized in independently permuted order as semicolon-separated coordinate–value pairs, with three-digit hexadecimal coordinates and two-digit hexadecimal values. The methods parse these records into the numeric array X by coordinate ID, so record order does not affect their input. Database values, record permutations, and query samples use separate NumPy PCG64 streams. Each stream is initialized from a SHA-256 hash of the seed, sampling purpose, database size, alphabet size, database ID, and, for queries, query law.

For each target law, coordinates are ranked by decreasing π_i, breaking ties by smaller ID, and divided into four equal-sized bins $G_0, \ldots, G_3$. Each database receives four queries sampled uniformly with replacement from each bin, giving 16 queries per database and $4{,}096$ test rows per seed, (m, q) pair, and query law. Consequently, test queries are marginally uniform over coordinates, while evaluation weights each query at coordinate i by $m\pi_i$ to estimate accuracy under the target query law. The balanced sampling also supports separate error estimates for each rank bin. All methods and budgets share the same test databases and query rows within a seed and query law, and the database files are shared across query laws.

Compared models and memory budgets. We evaluate fixed random literal retention, fixed top-s literal retention, a learned numeric summary, and, for $q = 2$, perfect-code summaries from Proposition 2.4 on the same benchmark. To compare them at equal logical memory budgets, let $v = \log_2 q$ be the number of bits per value and set $B = \lfloor \rho m v \rfloor$, where ρ is the retained fraction of the full database's bit count. The budget counts the meaningful bits retained between compression and query answering. Fixed model weights and public condition parameters are excluded. We use $\rho \in \{1/16, 1/8, 1/4, 1/2, 3/4, 1\}$. Together with the benchmark dimensions, query laws, and seeds, the budget sweep gives 648 conditions for each literal method and the neural summary, and 216 for the binary perfect-code summary. Each method receives the full database before the query, and each answer uses only the retained cache, the query, and fixed public parameters. Queries leave the cache and weights unchanged.

Both literal methods store $s = \rho m$ values, an integer throughout the grid. Random retention chooses a uniformly sampled size-s subset before reading the database. In contrast, top-s retention uses the known query law to choose the s most probable coordinates, breaking ties by smaller ID. Each retained set is public, independent of database values, and fixed within its condition. A retained query returns its exact value, while a miss returns a uniformly sampled alphabet value using a deterministic, method-specific random stream independent of database values. Neither literal method has trainable parameters.

The neural method encodes the database into a cache of B bits using learned codebooks. The decoder predicts a queried value using this cache and the fixed codebooks. The original database, encoder scores, and intermediate predictions are discarded at the compression boundary. Codebooks are trained separately for each condition and evaluated at the corresponding database size and alphabet.

Neural architecture. The neural model uses a grouped codebook architecture throughout the experiment. Each code index occupies $c = v \max\{1, \lfloor 4/v \rfloor\}$ bits, giving $c = 4$ for $q \in \{2, 16\}$ and $c = 6$ for $q = 64$. Because B is divisible by c throughout the grid, the database is partitioned into $J = B/c$ contiguous coordinate groups $\mathcal{I}_1, \ldots, \mathcal{I}_J$. Boundaries are chosen greedily to place the cumulative query mass after group j as close as possible to j/J, while ensuring that every group contains at least c/v coordinates. Thus, grouping depends on the public query law and memory budget, and remains fixed across databases.

Each group has an independent codebook with 2^c entries. For entry k and coordinate $i \in \mathcal{I}_j$, a softmax over q learned logits defines the value probabilities $p_{j,k,i}(a)$ for $a \in \{0, \ldots, q-1\}$. Given database X, the encoder chooses

$$k_j(X) = \arg\max_{0 \le k < 2^c} \sum_{i \in \mathcal{I}_j} \pi_i p_{j,k,i}(X_i).$$

The selected indices are packed using c bits per group, giving exactly B logical cache bits. For a query at coordinate $i \in \mathcal{I}_j$, the decoder reads the group's index and returns the most probable value under $p_{j,k_j(X),i}$. Ties in code selection and value prediction use the smallest index or value. A codebook entry can describe several coordinates jointly, allowing a retained index to summarize multiple records. The packed indices are the only database-dependent information retained for answering queries. The codebooks and group boundaries are fixed before testing.

Training and validation data. To fit the neural models, separate deterministic streams generate $32{,}768$ training databases and $2{,}048$ validation databases for each (m, q, seed). Values follow the same uniform independent sampling law as the test databases. Duplicate rejection makes each pool internally unique and the two pools mutually disjoint. The pools remain fixed and are reused across budgets and query laws. The model still adapts to the query law because the law determines group boundaries, weights code selection, and enters the training loss. Training databases supply the examples for optimizing weights, while validation databases select checkpoints. Both splits use every coordinate as a query, with the database value at that coordinate as its target.

Training process and checkpoint selection. We train one grouped codebook model for each combination of database size m, alphabet size q, budget fraction ρ, query law, and seed, giving 648 neural models across the full grid. The process seed controls initialization and the epoch-permutation stream. To initialize each codebook, its entries enumerate all value combinations on the first c/v coordinates of the group, while the remaining reconstruction symbols are sampled uniformly from the alphabet. The resulting one-hot vectors initialize the value logits.

Each epoch visits the shuffled training pool once and uses every coordinate, weighted by its target query probability π_i. Let $p_{d,i}(X_{d,i})$ be the probability assigned to the correct whole value by the selected codebook entry for coordinate i of training database d. For a batch of N_{batch} databases, the objective is

$$\mathcal{L} = \frac{1}{N_{\text{batch}}} \sum_d \sum_i \pi_i \left(1 - p_{d,i}(X_{d,i})\right).$$

Each optimizer step recomputes the hard code indices and holds them fixed when computing gradients of the selected codebook logits. Hard indices are used from the first epoch and during validation and testing, so the retained representation has the same precision throughout training and evaluation.

Training uses AdamW with initial learning rate 0.03, zero weight decay, and batch size 512. One complete epoch therefore contains 64 optimizer updates. The maximum training length is 150 epochs, with cosine learning-rate decay toward 0.003 over that schedule. After at least 60 epochs, training stops if validation accuracy has not strictly improved for 25 epochs. Validation enumerates all coordinates of every validation database with weights π_i. For each condition, the earliest checkpoint attaining the highest target-law validation accuracy is retained and frozen before test evaluation.

Accuracy and aggregation. For test row t, let i_t be its query coordinate and A_t indicate an exactly correct value prediction. Target-law accuracy uses the self-normalized importance weights $w_t = m\pi_{i_t}$,

$$\widehat{\mathrm{Acc}}_\pi = \frac{\sum_t w_t A_t}{\sum_t w_t}.$$

Bin-balanced accuracy is the unweighted mean over all test rows, which also equals the mean of the four bin accuracies. Tail accuracy is the accuracy in G_3, the least-probable rank bin, whereas worst-bin accuracy is the minimum across all four bins. For uniform queries, bins are determined by coordinate ID. Metrics are computed separately for each seed and then reported as the equal-weight mean and sample standard deviation over the three seeds, using denominator $3-1$ for the variance.

For each literal method, we compare target-law accuracy with the exact prediction from Corollary B.10,

$$\mathrm{Acc}_\pi^{\mathrm{lit}}(R) = \frac{1}{q} + \left(1 - \frac{1}{q}\right) \sum_{i \in R} \pi_i,$$

where R is the actual retained set. Random retention is compared with its realized set for each seed. Averaging over uniformly sampled sets gives $1/q + (1-1/q)s/m$. The learned summary can encode joint information about multiple coordinates, so the fixed-subset frontier is not an upper bound for that method.

Information bound. Following Theorem 2.2, define $f_q(e) = \log_2 q - h_2(e) - e \log_2(q-1)$, where h_2 is binary entropy. Let $\widehat{e}_b$ be the unweighted empirical error in rank bin G_b and $e_{U,b}$ its simultaneous nominal 99% upper bound. The point and conservative information estimates are

$$\widehat{S}_{\mathrm{Fano}} = \sum_{b=0}^{3} |G_b| f_q\left(\min\left\{\widehat{e}_b, 1 - \frac{1}{q}\right\}\right),$$

$$S^{99}_{\mathrm{Fano}} = \sum_{b=0}^{3} |G_b| f_q\left(\min\left\{e_{U,b}, 1 - \frac{1}{q}\right\}\right).$$

Using true bin errors gives the population bound $B \geq S_{\mathrm{Fano}}$. Indeed, the average MAP error in a bin is at most its population decoder error and at most c_q, so convexity of f_q gives this binned consequence of Theorem 2.2. The hatted errors are sampled estimates. Bootstrap upper bounds yield conservative information estimates on their coverage event, with the approximate coverage described below, and are distinct from worst-case guarantees.

Query-weighted information bound. For the population target-law error $\bar{e} = \sum_i \pi_i e_i$, the q-ary variational bound is

$$\mathcal{R}_\pi(\bar{e}) = \min_{\substack{0 \leq a_i \leq 1-1/q \\ \sum_{i=0}^{m-1} \pi_i a_i \leq \bar{e}}} \sum_{i=0}^{m-1} f_q(a_i),$$

where a_i is a candidate coordinate error and $B \geq \mathcal{R}_\pi(\bar{e})$. The minimum is computed by bisection, and its point estimate uses $\widehat{\bar{e}} = 1 - \widehat{\mathrm{Acc}}_\pi$. Because $\mathcal{R}_\pi$ is nonincreasing, a one-sided nominal 99% upper error bound $\bar{e}_U$ gives the conservative estimate

$$\mathcal{R}^{99}_\pi = \mathcal{R}_\pi(\bar{e}_U).$$

Both diagnostics use $10{,}000$ whole-database bootstrap resamples with base seed 172 and method- and condition-specific streams. Each resample retains all 16 queries per database and recomputes the bin errors and self-normalized query-weighted error, including its weight denominator. A maximum statistic across the four bin-error deviations gives $e_{U,b}$. $\bar{e}_U$ is the 99th percentile of the resampled

target-law errors, using the higher order statistic and constrained to be at least $\widehat{e}$. Coverage is approximate within each method and condition, not simultaneous across the full grid.

Fano frontier. The same information constraint defines an upper bound on target-law accuracy at each normalized memory budget ρ. Let a_i denote a hypothetical error probability for coordinate i. For the logical budget $B = \lfloor \rho m \log_2 q \rfloor$, the Fano frontier is

$$\mathrm{Acc}_{\mathrm{Fano}}(\rho) = 1 - \min_{\substack{0 \le a_i \le 1-1/q \\ \sum_{i=0}^{m-1} f_q(a_i) \le B}} \sum_{i=0}^{m-1} \pi_i a_i.$$

The minimization finds the smallest query-weighted error permitted by Theorem 2.2, so its complement bounds population accuracy for both literal retention and general summaries at every finite m. Under uniform queries this frontier is asymptotically achievable by part (c) of Theorem 2.2. Under the block-constant assumptions of Theorem B.6, the inverse variational function characterizes boundary memory up to $o(m)$ bits. For the Zipf laws used here, Corollary B.7 instead gives a blockwise achievable bound with an explicit allocation gap. The plotted unrestricted frontier remains a converse for those laws. The implementation computes the frontier by numerically inverting the minimum Fano information required for a given target-law error. The frontier depends on m, q, π, and ρ, but not on the process seed or observed test errors. It equals $1/q$ at zero budget and 1 at full budget.

Four-bin allocation gap. For the four rank bins $G_0, \ldots, G_3$ used above, let $D_F(B) = 1 - \mathrm{Acc}_{\mathrm{Fano}}(\rho)$. We evaluate the restricted variational problem in (19) at this same target error and define

$$\Delta_G(B) = \frac{\mathcal{R}_\pi^{(G)}(D_F(B)) - \mathcal{R}_\pi(D_F(B))}{m \log_2 q}. \tag{40}$$

Across the stated database sizes, alphabets, and budget fractions, its maximum is zero for uniform queries and approximately 6.08%, 10.09%, and 14.51% for Zipf exponents 0.8, 1.1, and 1.5, respectively. All three nonzero maxima occur at $m = 256$, $q = 64$, and $\rho = 1/16$. At these conditions, the available budget is $B = 96$ bits, whereas the restricted variational values are approximately 189.44, 250.93, and 318.82 bits. Normalizing the excess by the actual budget gives $[\mathcal{R}_\pi^{(G)}(D_F(B)) - \mathcal{R}_\pi(D_F(B))]/B$ of approximately 97.33%, 161.38%, and 232.10%, respectively. Thus, the restricted rates are about 1.97, 2.61, and 3.32 times the available budget, so the four-bin restriction is loose at these low-budget conditions. These values depend only on the query laws and the two variational problems. They measure the cost of imposing one target error per rank bin, before finite-block coding overhead, and are separate from sampled accuracy and bootstrap uncertainty.

C.2.2 Results

Figure 7 illustrates how the three methods distribute retrieval accuracy across query-rank bins for $m = 128$, $q = 16$, and $\rho = 1/4$, under four query laws. Random retention spreads its retained records across the database, giving roughly even accuracy across all bins. In contrast, Top-s retention focuses entirely on the first bin and leaves the other bins near chance, because its 32 retained coordinates exactly fill that bin. The grouped codebook learns a representation adapted to the query distribution. Under uniform queries, it distributes accuracy relatively evenly. As Zipf queries become more concentrated, it trades accuracy in the tail for higher accuracy in the head. Overall, none of the tested methods achieves high accuracy across all bins at this budget, and prioritizing the head comes at the expense of the tail.

Figure 1b compares the binned information lower estimates S^{99}_{Fano} with the logical budget B across methods, database sizes, alphabets, and query laws. All $1{,}944$ estimates satisfy $S^{99}_{\mathrm{Fano}}/B \le 1$, consistent with Theorem 2.2 for both literal retention and learned summaries. At fixed (m, q, ρ) and query law, Top-s gives larger seed-averaged ratios than Random. Top-s approaches one at $\rho \in \{1/4, 1/2, 3/4\}$, especially for $q \in \{16, 64\}$, because retaining whole rank bins makes the population Fano bound equal B. Random mixes exact recall with chance guesses within bins, so averaging their errors reduces the Fano sum by convexity. Across query laws, literal-retention ratios remain similar because the coordinate ranking is unchanged and the binned diagnostic does not weight errors by π_i. The grouped codebook usually exceeds Random and generally yields higher ratios under stronger Zipf concentration, because its query-weighted training improves head-bin reconstruction and increases those bins' Fano contributions. Across methods, ratios generally increase

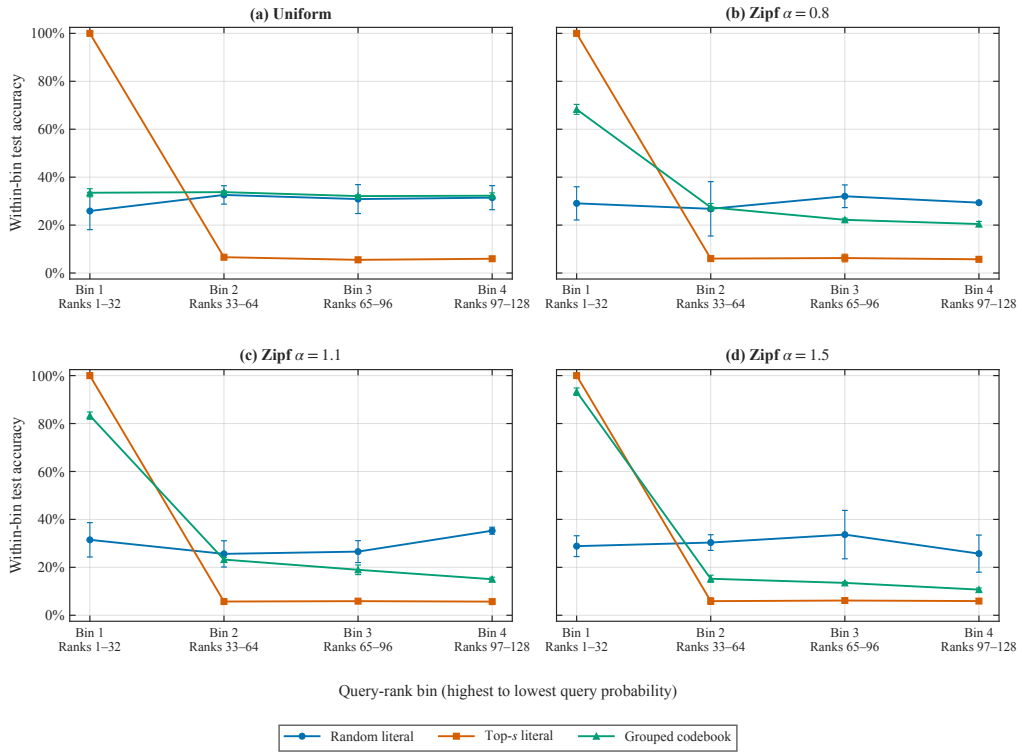


Figure 7: Test accuracy across query-rank bins for the Random and Top-s literal retention methods and the grouped codebook neural model, with database size $m = 128$ and alphabet size $q = 16$, at budget fraction $\rho = 1/4$ and logical cache budget $B = 128$ bits. Each condition and seed uses 256 test databases with 16 queries per database, giving 4,096 query occurrences in total. Accuracy is the unweighted fraction of correct answers within each bin, using 1,024 queries per bin for each condition and seed. Points show equal-weight means across seeds 47, 202, and 708. Error bars show $\pm$ 1 sample standard deviation.

with q, consistent with the smaller normalized entropy penalty $h_2(e)/\log_2 q$ at a given error e for larger alphabets. The dependence on m is comparatively weak at fixed ρ, because both the Fano sum and $B = \rho m \log_2 q$ scale linearly with m for fixed bin errors. At full retention, all three methods recover every tested value and the ratio reaches one, matching the theorem's zero-error endpoint $B = m \log_2 q$.

Figure 8 compares the query-weighted ratio $\mathcal{R}_\pi^{99}/B$ across the same conditions. All estimates remain at or below one, consistent with Theorem 2.2. Under uniform queries, Random and Top-s yield similar ratios because Corollary B.10 gives them the same population accuracy, and $\mathcal{R}_\pi$ depends on their average error. The grouped codebook often gives larger ratios, particularly for $q = 2$, reflecting its lower average error. Under Zipf queries, Top-s and the grouped codebook both exceed Random because their higher target-law accuracies tighten the variational error constraint, increasing the information required to support that accuracy. Random's ratio generally falls as queries become more concentrated, because the variational bound permits larger errors on rarely queried coordinates while maintaining comparable average accuracy. As in Figure 1b, the grouped codebook's ratio generally rises with Zipf concentration, the literal-retention ratios increase with q, and all methods reach one at full retention. The grouped codebook has weaker dependence on q, consistent with its larger accuracy gains over literal retention for binary records. At fixed ρ, the weak dependence on m under uniform queries also matches the binned diagnostic. Under Zipf queries with $\alpha = 1.5$, however, increasing m reduces the Top-s and grouped codebook ratios. The decrease partly reflects a larger bootstrap reduction from the point bound, consistent with fewer observations of each high-probability coordinate at the fixed test size. Together, the method and query-law trends

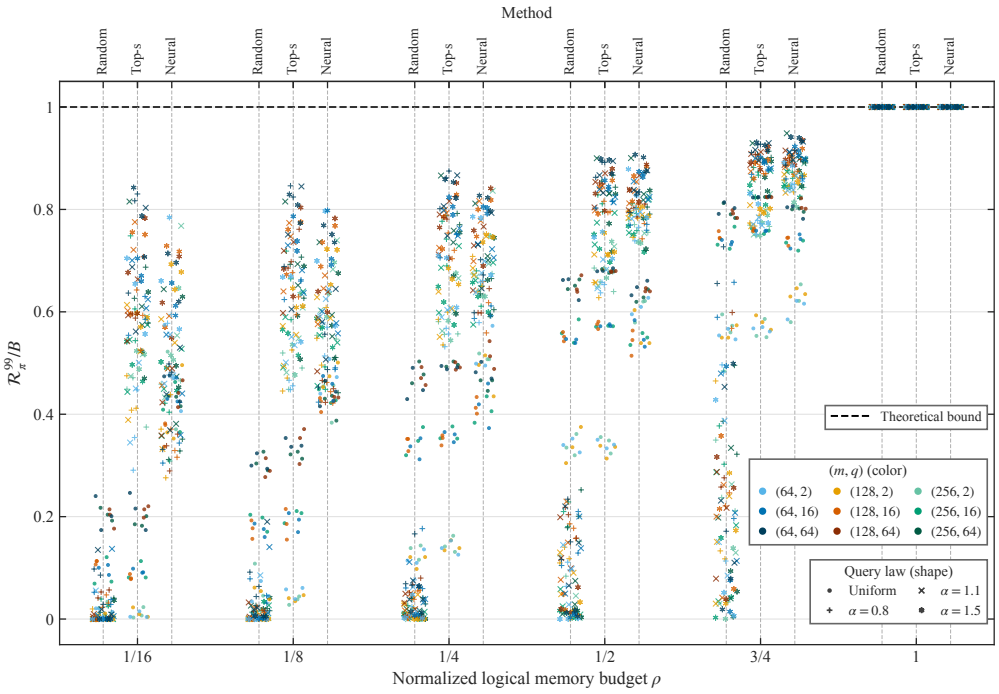


Figure 8: Estimated query-weighted Fano information relative to the logical cache budget for the Random and Top-s literal retention methods and the grouped codebook neural model across all conditions. Each condition and seed uses 256 test databases with 16 queries per database, giving 4,096 query occurrences in total. Following Theorem 2.2, the horizontal dashed line at $\mathcal{R}_\pi^{99}/B = 1$ marks the theoretical budget threshold. Estimates should lie on or below this line when the bootstrap confidence bounds hold. Points are individual estimates for seeds 47, 202, and 708, without averaging across seeds. Each estimate uses 10,000 whole-database bootstrap resamples with base seed 172 and method- and condition-specific streams to obtain a one-sided nominal 99% upper bound on the query-weighted error.

support the theorem's relation between average retrieval error, query probabilities, and necessary memory.

Figures 9 to 11 show accuracy versus ρ, with database sizes in rows and query laws in columns. Accuracy generally rises with ρ and falls with q. For literal retention, these trends agree with the prediction $1/q+(1-1/q)\pi(R)$ in Corollary B.10, because a larger retained query mass improves recall while a larger alphabet reduces the chance of a correct guess. Under uniform queries, Random and Top-s have nearly identical accuracy because $\pi(R) = \rho$, independently of m. The grouped codebook achieves its largest gains for binary records, exceeding both literal methods in seed-averaged accuracy at every tested $\rho < 1$ under uniform queries. Corollary B.9 gives the asymptotic counterpart of this alphabet dependence under uniform queries, with an advantage over literal retention of order $1/\log_2 q$ at fixed budget fraction. Joint encoding lets each code index support partial recovery across several coordinates, explaining the gains over literal storage. Across methods, the weak dependence on m under uniform queries is consistent with the normalized bound $\rho \log_2 q \geq f_q(\bar{e})$ from Theorem 2.2, since keeping ρ fixed makes the budget $B = \rho m \log_2 q$ grow linearly with m. Under Zipf queries, Top-s and the grouped codebook become more accurate as query concentration or m increases at fixed ρ. Both methods favor the leading ranks, whose share of query mass increases in these settings, while Random's expected retained mass remains ρ under every query law. The learned preference for frequent coordinates also agrees with the theorem's binary variational optimizer, which assigns lower error to more probable queries. Across the full grid, seed-averaged accuracies remain at or below the Fano frontiers derived from Theorem 2.2, consistent with the theoretical memory–accuracy tradeoff for both literal retention and learned summaries.

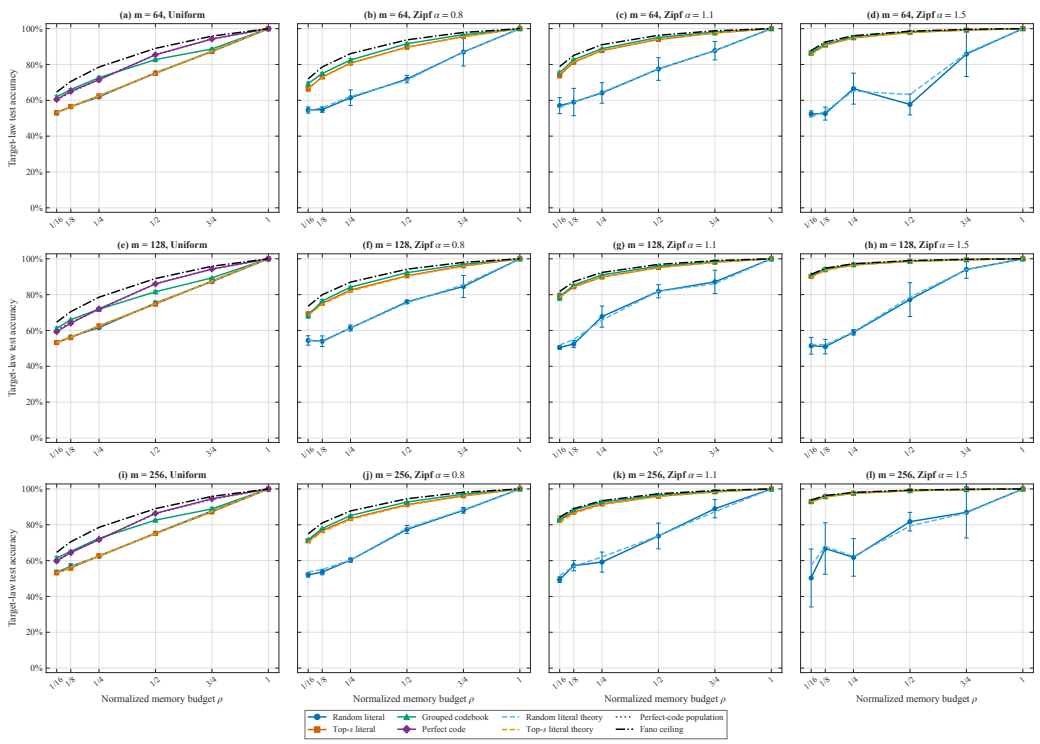


Figure 9: Target-law test accuracy versus normalized logical memory budget ρ for the Random and Top-s literal retention methods, perfect codes summaries and the grouped codebook neural model, with binary records, $q = 2$. Each condition and seed uses 256 test databases with 16 queries per database, giving 4,096 query occurrences in total. Points show equal-weight means across seeds 47, 202, and 708, with error bars indicating $\pm$ 1 sample standard deviation. Light blue and gold dashed curves show the mean theoretical accuracies for the realized random and top-s retained sets from Corollary B.10. The black dash-dot curve is the Fano frontier derived from Theorem 2.2, which shows the theoretical upper bound.

The uniform-query panels of Figure 9 compare the learned summary with attainable perfect-code compositions at the same budget B. At $m = 128$ and $B = 64$, Proposition 2.4 gives 86.12% population accuracy, compared with 75% for literal retention. Evaluation on the same test databases gives perfect-code accuracy $86.00 \pm 0.44\%$ and grouped-codebook accuracy $81.58 \pm 0.83\%$, with means and sample SDs across three seeds. The gain over literal retention arises because coded blocks preserve partial information about several records jointly. However, the grouped codebook has higher empirical mean accuracy at $\rho \in \{1/16, 1/8\}$ for every tested m, so the comparison depends on the budget. At compressed budgets, the dotted purple population curve remains below the Fano frontier from Theorem 2.2, separating attainable performance within this family from the converse bound on arbitrary summaries. Optimizing this code family does not establish finite-length optimality.

To examine allocation beyond the four rank bins, we pool saved error counts and query counts across three seeds for each coordinate at $q = 2$, $\rho = 1/4$, and $m \in \{64, 128\}$. We compare these empirical errors with the optimizer defining the Fano frontier in Theorem 2.2 at the same budget. Under uniform queries, the optimizer assigns error 21.45% to every coordinate, while empirical errors vary around a higher average. Under Zipf queries with $\alpha = 1.1$, empirical errors generally increase toward the tail, consistent with the optimizer's preference for frequent queries because the training objective weights coordinates by their query probabilities. At $m = 128$, the mean absolute coordinate deviation from the optimizer is 6.99 points under uniform queries and 6.62 under Zipf queries. These descriptive estimates support the predicted allocation pattern without establishing that the finite codebook realizes the converse optimizer.

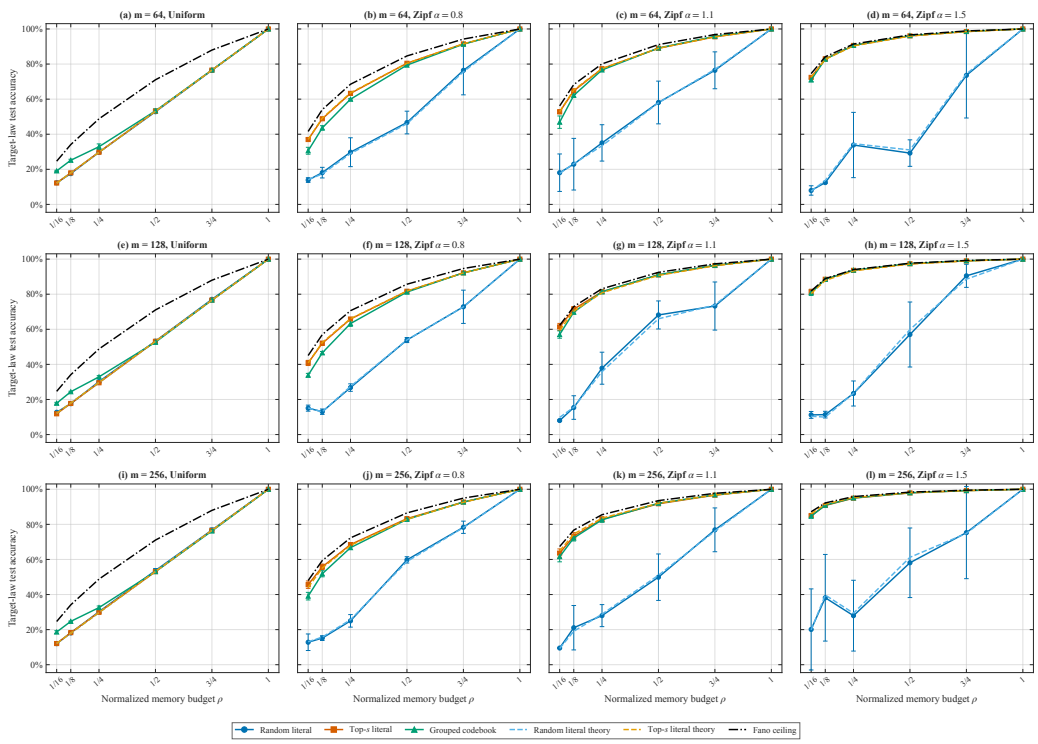


Figure 10: Target-law test accuracy versus normalized logical memory budget ρ for the Random and Top-s literal retention methods and the grouped codebook neural model, with alphabet size $q = 16$. Each condition and seed uses 256 test databases with 16 queries per database, giving 4,096 query occurrences in total. Points show equal-weight means across seeds 47, 202, and 708, with error bars indicating $\pm$ 1 sample standard deviation. Light blue and gold dashed curves show the mean theoretical accuracies for the realized random and top-s retained sets from Corollary B.10. The black dash-dot curve is the Fano frontier derived from Theorem 2.2, which shows the theoretical upper bound.

C.3 Experiment 3: Query timing and KV-cache compression

The task is the same associative recall problem as in Experiment 2: given a database of indexed values and a query coordinate i, recover the corresponding value X_i. Here, a frozen language model reads the serialized database and generates the requested value using its retained key–value (KV) cache. The experiment examines how retrieval changes when the query becomes available before or after cache compression. The motivation is that a cache prepared for unknown future queries must preserve information useful across coordinates, whereas a query-dependent cache can concentrate on the current request. Under a fixed query law, pre-query compression produces a reusable state $C(X)$, matching the information flow in part (a) of Theorem 2.1 and the converse in Theorem 2.2. Processing the query before compression instead permits a state $C(X, i)$, so the query-independent premise of the coordinatewise converse no longer applies at that boundary. Part (b) of Theorem 2.1 formalizes the corresponding exact-recall budgets. We compare these processing orders under the same frozen model and examine both retrieval accuracy and the state retained at each boundary.

C.3.1 Settings

Test benchmark. We reuse the saved benchmark described in Appendix C.2.1, including independent uniform record values, hexadecimal serialization, randomly permuted record order, database sizes $m \in \{64, 128, 256\}$, alphabet sizes $q \in \{2, 16, 64\}$, and the uniform and three Zipf query laws. The evaluated cohort uses the first 64 databases for each (m, q) pair under seed 47. Each database has the same 16 saved queries per law, with four samples drawn uniformly with replace-

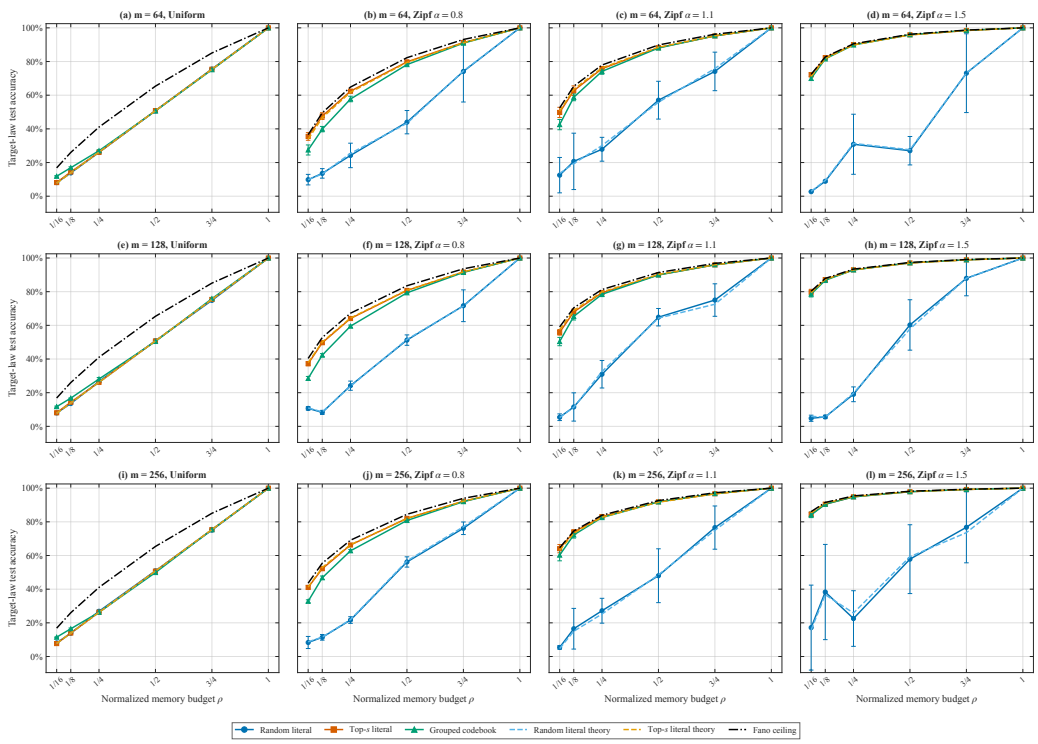


Figure 11: Target-law test accuracy versus normalized logical memory budget ρ for the Random and Top-s literal retention methods and the grouped codebook neural model, with alphabet size $q = 64$. Each condition and seed uses 256 test databases with 16 queries per database, giving 4,096 query occurrences in total. Points show equal-weight means across seeds 47, 202, and 708, with error bars indicating $\pm$ 1 sample standard deviation. Light blue and gold dashed curves show the mean theoretical accuracies for the realized random and top-s retained sets from Corollary B.10. The black dash-dot curve is the Fano frontier derived from Theorem 2.2, which shows the theoretical upper bound.

ment from each of the four query-rank bins. Thus, each (m, q, π) condition contains 1,024 query occurrences.

Prompt construction and processing orders. Every request uses one native user-message wrapper and the following three text blocks. Italic placeholders denote the saved serialized database and the three-digit hexadecimal query coordinate.

1. **Database block:**

   ```
   Remember this KEY=VALUE database.
   #Database:
   ```
   [serialized database]

2. **Query block:**

   ```
   Find the KEY=VALUE entry for this key in the database.
   KEY:
   ```
   [query coordinate]

3. **Output block:**

   ```
   Copy only its two-digit VALUE.
   #Answer:
   ```

Headers, complete record spans, and query components are tokenized separately without internal special tokens, so physical record removal can retain complete spans. We use three processing orders.

1. *Pre-query compression.* The model first processes the database block and constructs a cache before seeing the query. Compression occurs at the end of that block. A private copy of the retained cache then processes the query and output blocks before generating an answer. The same database cache is reused for all 16 independent requests.
2. *Post-query compression.* The model processes the database block followed by the current query block, then compresses the resulting cache. It subsequently processes the output block and generates the answer. The retained state can therefore depend on the query, although the database was originally processed without it.
3. *Query-first compression.* The query block precedes the database block, and compression occurs after the database has been processed. Consequently, both the database representations and the retained state can depend on the query. The output block follows compression, and the query appears only once.

The native message closing tokens follow the output block in every order. Therefore, the first answer-token logits are computed after compression by processing the remaining prompt against the retained cache. No answer logits computed from the uncompressed prefix survive the compression boundary.

Frozen model and decoding. All neural conditions use Mistral AI's Ministral 3 3B Instruct 2512 checkpoint, pinned to revision `b35d4dfe56c1`. Its text decoder has 26 layers, 32 query heads, and 8 KV heads per layer, with head dimension 128. Projection weights and activations use the checkpoint's native FP8 format and static scales, with FP32 accumulation and BF16 outputs. Embeddings, normalization, attention, residual states, and the output head remain BF16. Every cache method stores BF16 keys and values. The model is frozen throughout evaluation, with no task-specific training or validation pool. Decoding is greedy, stopping at an end-of-sequence token or after 8 generated tokens. Prefill uses chunks of at most 1,024 tokens, and the complete prompt plus the answer reservation fit within 32,768 tokens.

Cache methods and retention settings. Full KV retains every prefix position and provides a separate uncompressed reference for each processing order. Compressed conditions use retention fractions $\rho \in \{1/8, 1/4, 1/2, 3/4\}$. Physical record removal is evaluated only with pre-query compression, while SnapKV is evaluated under all three orders. The grid therefore has 13 pre-query settings and 5 settings for each later order, giving 23 neural settings per benchmark condition.

For *physical Top-s removal*, the model first processes the entire database and then retains the KV entries for the $s = \lfloor \rho m \rfloor$ most probable query coordinates, breaking ties by smaller coordinate ID. *Physical Random removal* retains the first s coordinates of a uniformly random permutation fixed by the seed, m, q, and query law. Using one permutation across fractions makes the random retained sets nested. Both methods preserve the public prompt header and every token in each selected record span, with the same selected records across layers and heads. Because the surviving KV vectors were computed while reading the full database, they can contain information about earlier records whose own KV entries are removed.

SnapKV (Li et al., 2024) selects token positions using attention from the final 32 tokens of the prefix available at compression. If that prefix has n tokens, the retained capacity per KV head is

$$k = \min\{n, \max\{33, \lceil \rho n \rceil\}\}.$$

The implementation sums causal attention probabilities over the observation window, averages the four query heads sharing each KV head, and applies average pooling of width 5 to the historical scores. Each layer and KV head retains the $k - 32$ highest-scoring historical positions together with the full observation window. Ties favor earlier positions. The prefix contains only the database block in the pre-query order and both database and query blocks in the later orders. Selection therefore uses no query in the first case and can use query-dependent states in the latter two. No additional query or header positions are protected outside this capacity. All retained entries preserve their original positional indices. After selection, the remaining prompt and generated tokens append KV entries without further pruning or reselection.

Scoring method. A generated answer is scored as correct when the two-digit reference value appears anywhere in the decoded response, using case-sensitive substring matching without answer extraction. A response lacking that value is an error. Target-law accuracy uses the same self-

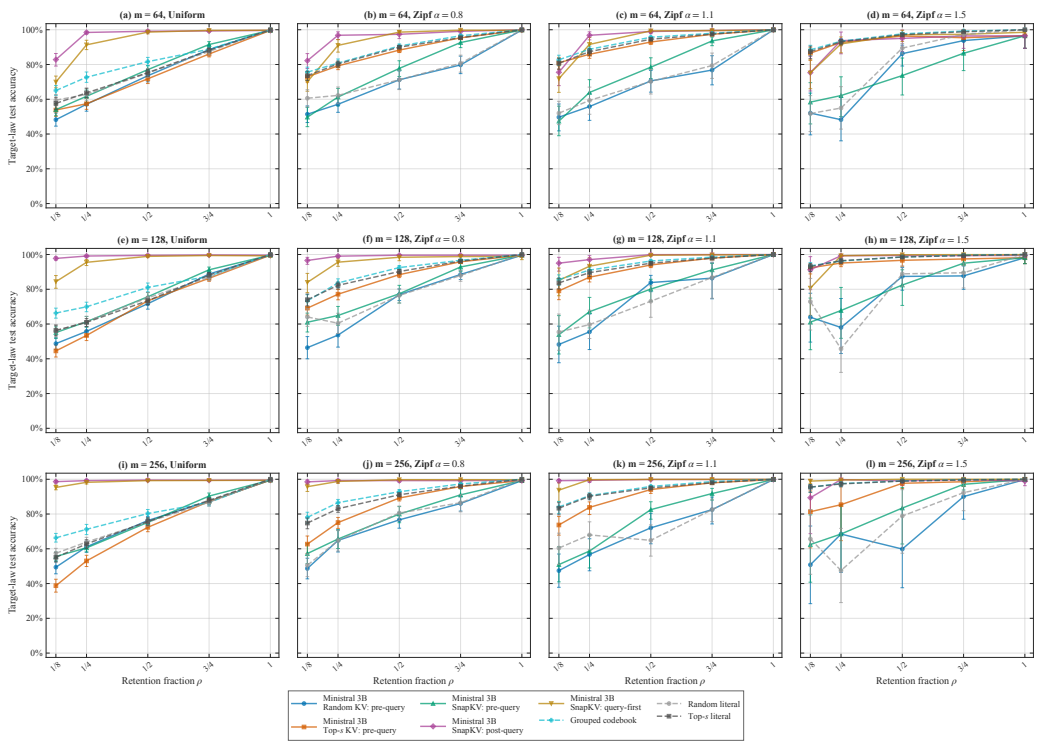


Figure 12: Target-law test accuracy versus retention fraction ρ for binary records, $q = 2$. Solid curves show frozen Ministral 3 3B Instruct 2512 with pre-query Random KV removal, pre-query Top-s KV removal, and SnapKV under pre-query, post-query, and query-first processing. Dashed curves show measured accuracy for grouped codebook, Random literal retention, and Top-s literal retention. Each condition uses 64 matched databases and 1,024 query occurrences under data seed 47. Error bars are 95% percentile confidence intervals from 10,000 paired bootstrap resamples of whole databases, using base bootstrap seed 172. The fraction ρ refers to records for physical and literal retention, prefix-token capacity for SnapKV, and logical memory for grouped codebook.

normalized weights $w_t = m\pi_{i_t}$ as Experiment 2. Bin-balanced accuracy is the unweighted mean, and worst-bin accuracy is the minimum accuracy across the four query-rank bins.

We report gains as differences in target-law accuracy on matched databases and query occurrences, expressed in percentage points. For each setting, we draw 10,000 paired bootstrap resamples of 64 whole databases with replacement, using base bootstrap seed 172. Both methods use the same sampled databases, with all query occurrences and their weights retained. The 2.5th and 97.5th percentiles of the resampled accuracy differences give the 95% confidence interval. Each interval describes one setting without adjustment for multiple comparisons; the median summarizes observed gains across settings.

C.3.2 Results

Across Figures 12 to 14, the solid Random and Top-s KV curves broadly follow their dashed literal controls. The two physical methods achieve similar accuracy under uniform queries, while Top-s KV performs better under Zipf queries. The pattern agrees with Corollary B.10, with literal accuracy depends on the total query probability of the retained records, which Top-s maximizes. At fixed ρ, increasing m also increases the query probability captured by the leading records under Zipf $\alpha = 0.8$ and 1.1, explaining the shared size gains of physical and literal Top-s. Random KV varies little overall with m. Across the grid, all five LLM methods improve with ρ and perform best on binary records on average, consistent with retaining more context and reducing answer uncertainty, respectively. As more records are retained, Top-s KV's average gap from its literal control narrows from 5.7 to 1.5 percentage points between $\rho = 1/8$ and $3/4$. Under concentrated queries, Top-s KV

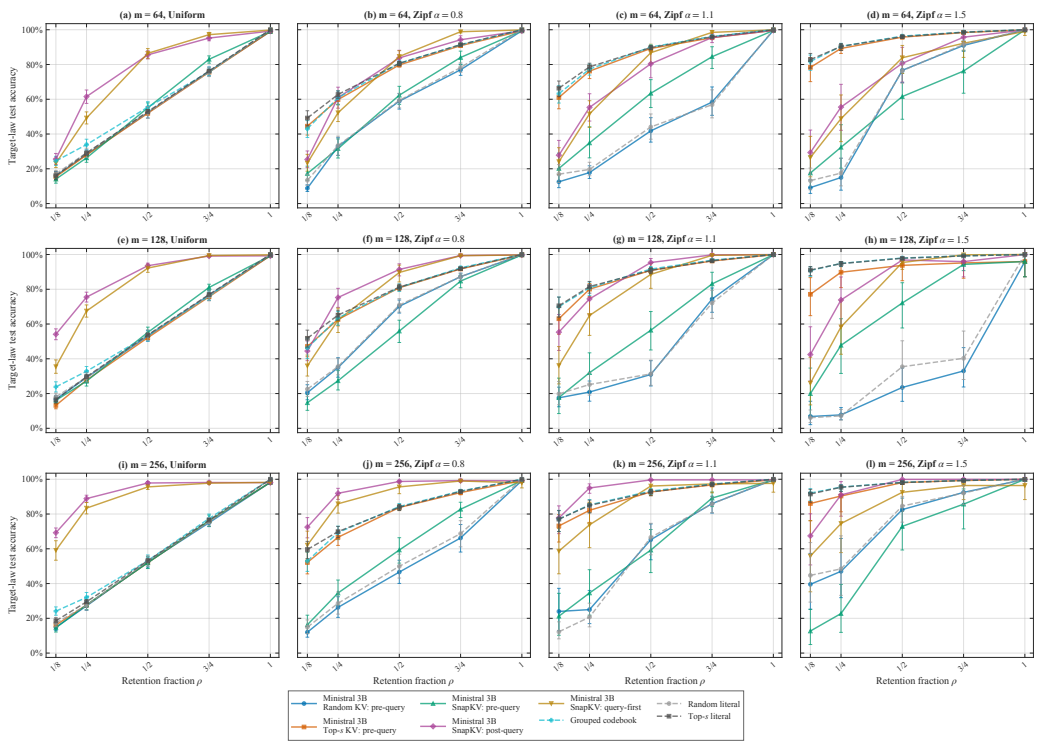


Figure 13: Target-law test accuracy versus retention fraction ρ for alphabet size $q = 16$. Solid curves show frozen Ministral 3 3B Instruct 2512 with pre-query Random KV removal, pre-query Top-s KV removal, and SnapKV under pre-query, post-query, and query-first processing. Dashed curves show measured accuracy for grouped codebook, Random literal retention, and Top-s literal retention. Each condition uses 64 matched databases and 1,024 query occurrences under data seed 47. Error bars are 95% percentile confidence intervals from 10,000 paired bootstrap resamples of whole databases, using base bootstrap seed 172. The fraction ρ refers to records for physical and literal retention, prefix-token capacity for SnapKV, and logical memory for grouped codebook.

also often approaches grouped codebook accuracy. Their shared emphasis on frequently queried records agrees with the query-weighted perspective of Theorem 2.2, whose binary optimizer assigns lower error to more probable queries.

Pre-query SnapKV broadly resembles Random retention and varies little overall with m. Across 144 tested settings, post-query and query-first SnapKV have median accuracy gains over pre-query SnapKV of 26.80 and 20.88 percentage points, respectively. The confidence intervals lie entirely above zero in 139 and 133 settings. Under uniform queries, their median gains over literal Top-s are 30.91 and 22.56 percentage points across 36 settings, with intervals entirely above zero in 35 settings for each method. Because all records are equally likely to be requested, the gains illustrate the benefit of using the actual query to select relevant information. Both variants generally improve with m, which partly reflects the larger absolute token capacity available at fixed ρ. The timing advantage agrees qualitatively with part (b) of Theorem 2.1. For exact deterministic recall in the abstract model, compression before the query requires $\lceil m \log_2 q \rceil$ bits to support every possible request, whereas $\lceil \log_2 q \rceil$ bits suffice at the boundary after both blocks because only the requested value must survive. Both query-aware SnapKV orders compress after processing the database and query, allowing selection to prioritize evidence for that value. The observed gains therefore illustrate the explanatory value of Theorem 2.1, which identifies the cache with a one-way message and connects communication requirements to the information needed at each processing boundary. These timing budgets motivate making the query available before compression when the retained state serves a single request. For literal-only state, Proposition 2.3 predicts exact agreement with the literal line if selection is value-independent, and bounds the possible excess by ϵ_{sel} otherwise. Example B.11

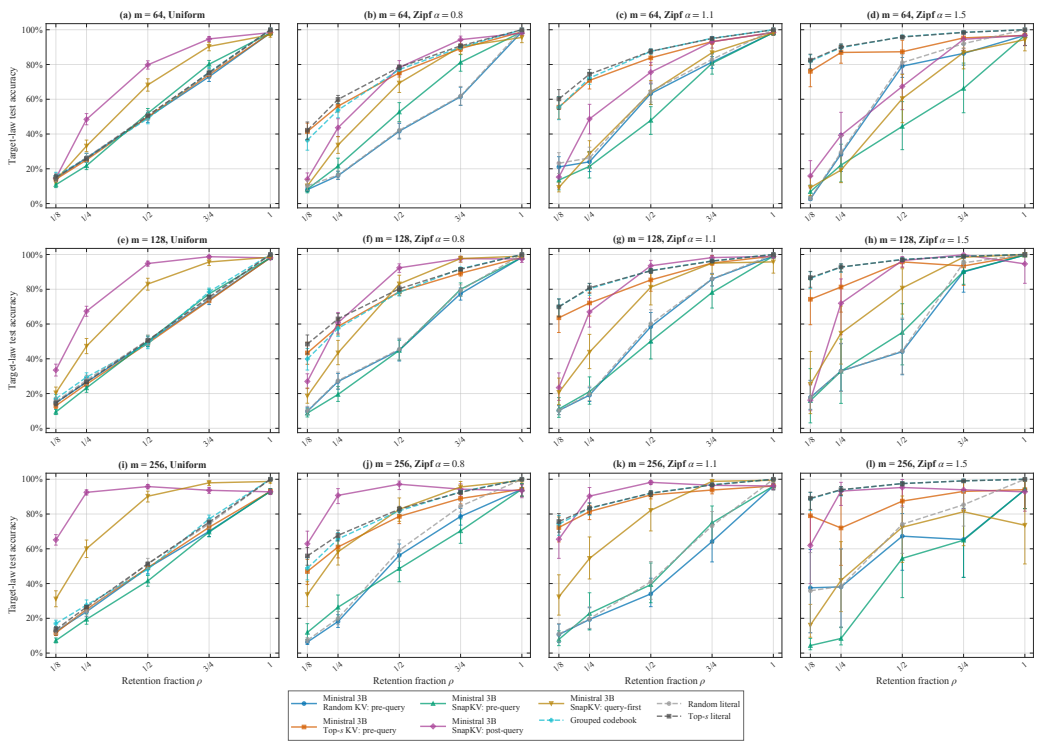


Figure 14: Target-law test accuracy versus retention fraction ρ for alphabet size $q = 64$. Solid curves show frozen Ministral 3 3B Instruct 2512 with pre-query Random KV removal, pre-query Top-s KV removal, and SnapKV under pre-query, post-query, and query-first processing. Dashed curves show measured accuracy for grouped codebook, Random literal retention, and Top-s literal retention. Each condition uses 64 matched databases and 1,024 query occurrences under data seed 47. Error bars are 95% percentile confidence intervals from 10,000 paired bootstrap resamples of whole databases, using base bootstrap seed 172. The fraction ρ refers to records for physical and literal retention, prefix-token capacity for SnapKV, and logical memory for grouped codebook.

shows how value-dependent selection can encode additional information through the retained indices. The present comparison does not estimate that dependence or establish that contextual KV entries are literal records.

Table 4 shows that a fixed retention fraction does not imply the same complete memory ratio across methods or database sizes. Position buffers and the remaining prompt contribute additional state, so the accounting required by Remark B.14 includes more than retained KV values. Post-query SnapKV also keeps the full prefix for later requests, while query-first processing requires a new database prefill. Their smaller answer-time allocations therefore concern a known query, consistent with part (b) of Theorem 2.1 and Corollary B.12, which distinguish a known-query boundary from the information retained for future requests.

C.4 Experiment 4: Retrieval and causal use of long-context caches

In this experiment, we study how a frozen language model uses its key–value (KV) cache to answer retrieval queries about a long shared context in SCBench (Li et al., 2025). A request may ask for the value associated with a key, a dictionary word matching a prefix and suffix, or the variables linked to a numerical value through assignment chains. Each context supports five requests, which are evaluated independently. We examine both whether retrieval performance is preserved under cache reduction and whether the retained cache entries for a queried record causally affect the answer.

Retrieval accuracy alone, however, does not identify which retained state supports an answer. A record's own KV entries may be removed while information about that record remains elsewhere in

Table 4: Experiment 3 complete allocated state, in MiB (2^{20} bytes). Ranges span all evaluated database sizes, alphabets, query laws, and saved request footprints. Pre, post, and first denote the three compression orders. KV and Pos. give answer-time payload and position buffers; each Answer total also includes 262,392 bytes for logits and controller state. Quantization metadata and residuals are zero. Reuse is the prefix retained for later requests; – requires a new database prefill. A and R give percentages of the matched Full-KV answer and prefix states. Range endpoints need not coincide across columns. Public weights and temporary construction memory are excluded.

Method / order	ρ	KV	Pos.	Answer	Reuse	A (%)	R (%)
Full KV (pre/post)	1	51.0–183.8	0.100–0.359	51.3–184.4	46.8–179.9	100.0	100.0
Full KV (first)	1	51.0–183.8	0.100–0.359	51.3–184.4	–	100.0	–
Random (pre)	1/8	11.2–28.3	0.022–0.055	11.4–28.6	6.9–24.1	14.9–22.5	12.8–15.0
	1/4	16.9–50.7	0.033–0.099	17.1–51.0	12.6–46.5	27.1–33.6	25.3–27.2
	1/2	28.2–95.4	0.055–0.186	28.5–95.8	24.0–91.3	51.3–55.8	50.1–51.5
	3/4	39.6–139.9	0.077–0.273	39.9–140.4	35.4–135.9	75.7–78.0	75.1–75.9
Top-s (pre)	1/8	11.2–28.3	0.022–0.055	11.4–28.6	6.9–24.1	15.4–22.5	13.3–15.0
	1/4	16.9–51.1	0.033–0.100	17.1–51.4	12.6–46.9	27.8–33.6	26.0–27.2
	1/2	28.2–96.6	0.055–0.189	28.5–97.0	24.0–92.5	52.5–55.8	50.7–51.5
	3/4	39.6–140.9	0.077–0.275	39.9–141.4	35.4–136.9	76.5–78.0	75.3–76.1
SnapKV (pre)	1/8	10.2–26.7	0.159–0.417	10.6–27.4	6.0–22.8	14.8–20.6	12.7–12.8
	1/4	15.9–49.2	0.249–0.768	16.4–50.2	11.9–45.6	27.2–32.0	25.3
	1/2	27.6–94.0	0.432–1.469	28.3–95.8	23.7–91.2	51.9–55.1	50.7
	3/4	39.3–138.9	0.614–2.171	40.2–141.4	35.6–136.8	76.6–78.3	76.0
SnapKV (post)	1/8	8.3–24.9	0.130–0.389	8.7–25.5	46.8–179.9	13.8–17.0	100.0
	1/4	14.4–47.6	0.225–0.744	14.9–48.6	46.8–179.9	26.3–29.0	100.0
	1/2	26.6–93.0	0.416–1.454	27.3–94.7	46.8–179.9	51.3–53.1	100.0
	3/4	38.8–138.4	0.606–2.163	39.7–140.8	46.8–179.9	76.3–77.2	100.0
SnapKV (first)	1/8	8.3–24.9	0.130–0.389	8.7–25.5	–	13.8–17.0	–
	1/4	14.4–47.6	0.225–0.744	14.9–48.6	–	26.3–29.0	–
	1/2	26.6–93.0	0.416–1.454	27.3–94.7	–	51.3–53.1	–
	3/4	38.8–138.4	0.606–2.163	39.7–140.8	–	76.3–77.2	–

the cache. Therefore, *Part A* measures retrieval as the physical cache state is reduced, while *Part B* replaces the queried record's KV entries between matched contexts and measures the resulting answer changes.

C.4.1 Part A: Behavioral retention settings

Test benchmark. We use the String Retrieval tasks from SCBench (Li et al., 2025), with the dataset pinned to revision `283310bb8c5b`. KV Retrieval asks for a specified key's value among explicit records. Prefix-Suffix Matching asks for the latest dictionary entry satisfying both string constraints. Variable Tracking asks for all variables connected to a requested value through copy assignments. The release contains 100, 100, and 90 shared-context sessions for these tasks, respectively, with five queries per session. We only use 30 sessions per task, giving 90 distinct contexts and 450 queries per cache condition. The subset was fixed by deterministic hash selection within task-specific length strata, and its session identities are preserved across methods. The cohort includes the initial model-selection pilots. We therefore treat its results as a descriptive comparison of the fixed cache conditions.

The selected-model tokenization audit covers all 290 released sessions and finds no length exclusions at the operational limit of 172,032 tokens, including the answer allowance. Inputs are not truncated. Within the evaluated subset, shared prefixes contain 170,397–170,906 tokens for KV Retrieval, 117,345–117,717 for Prefix-Suffix Matching, and 125,067–125,092 for Variable Tracking.

Frozen model and decoding. Both parts use Ministral 3 8B Instruct 2512, pinned to revision `5b26027e7b19`, with its native tokenizer, chat template, and positional scaling. The text decoder has 34 layers, 32 query heads, and 8 KV heads per layer, with head dimension 128. The published FP8 checkpoint is dequantized once to BF16 during loading, and every cache condition then uses the same frozen BF16 decoder. Full and position-selected caches store BF16 keys and values. No weights or adapters are trained, and there are no task-specific training or validation pools. Evaluation uses batch size one with prefill chunks of 2,048 tokens. Decoding is greedy and stops at an end-of-sequence token or after 512 generated tokens, including the end-of-sequence token when present.

Prompt and independent requests. The prompt preserves the released context and query text. KV Retrieval and Prefix-Suffix Matching request only a "`FINAL:`" answer. Variable Tracking additionally asks the model to trace the relevant assignment chain forward before giving its answer. Only the text following the last final-answer marker is scored. The shared token prefix ends before query-specific tokens, and concatenating the saved prefix and request suffix reproduces the complete formatted prompt. Each request starts from a private branch of the same method-specific prefix state. Previous queries, answers, and request-specific cache updates do not enter later requests.

Cache methods, budgets, and continuation. We evaluate 15 conditions comprising one Full-KV reference, four budgets each for StreamingLLM and the two SnapKV variants, and two KIVI precisions. Position-removal methods use retention fractions $\rho \in \{1/8, 1/4, 1/2, 3/4\}$. For n available token positions, let k denote the retained count, including protected positions, and $k_{\min}$ the method-specific minimum capacity. We set $k = \min\{n, \max\{k_{\min}, \lceil \rho n \rceil\}\}$.

Full KV retains every prefix position in BF16 as the uncompressed reference. All methods except query-aware SnapKV construct reusable pre-query states. Query processing and generation then use only the retained method-specific state.

StreamingLLM (Xiao et al., 2024) retains four initial attention-sink positions and the $k - 4$ most recent positions, with $k_{\min} = 5$. Each budget uses a separate streamed prefill with the corresponding causal window, continues eviction during query processing and generation, and preserves original RoPE positions.

SnapKV (Li et al., 2024) uses the same token-selection rule and hyperparameters as Experiment 3 (Appendix C.3.1). Each layer and KV head retains the final 32 observation tokens and the $k - 32$ highest-scoring historical positions, with $k_{\min} = 33$. Frozen SnapKV selects once at the shared-prefix boundary. In contrast, query-aware SnapKV selects separately after processing each query and prompt terminator against Full KV, so n includes both prefix and query tokens. The first-token logits are preserved from before selection. Therefore, the first answer token benefits from Full KV. We record both the compressed answer-time state and the full prefix retained for future requests. Both variants append subsequent KV entries without reselection.

KIVI (Liu et al., 2024) quantizes the full BF16 prefix to 4- or 2-bit keys and values while retaining all positions. Keys use per-channel groups of 32 tokens, values use per-token groups of 32 channels, and the residual-length parameter is 32. Residuals, scales, and minima remain BF16, with codes packed into 32-bit words. Continuation uses asymmetric residual updates and layerwise BF16 reconstruction for attention.

Scoring method. The official SCBench scorer is pinned to revision `29ef1974fcb5`. KV Retrieval and Prefix-Suffix Matching score whether the reference string occurs in the extracted answer. Variable Tracking gives the rounded fraction of reference variable names found, ignoring case. A missing final-answer marker yields an empty extracted answer and a zero score. A score of one denotes full correctness for all three tasks.

For each task–budget comparison, we average the five query scores within each context and measure the mean score difference between query-aware and frozen SnapKV in percentage points. We draw 10,000 paired bootstrap resamples of the 30 whole contexts with replacement, using bootstrap seed 172 and the same sampled contexts for both methods. The 2.5th and 97.5th percentiles of the resampled score differences give the 95% confidence interval for each comparison, without adjustment for multiple comparisons.

C.4.2 Part A: Behavioral retention results

Part A extends the controlled retrieval experiments to LLM KV-cache compression on SCBench, an established long-context benchmark. Table 1 compares 15 cache conditions for frozen Ministral 3 8B across KV Retrieval (KV), Prefix-Suffix Matching (PS), and Variable Tracking (VT). Increasing ρ improves all three task scores for StreamingLLM and both SnapKV variants as more context remains available to the continuation, extending the memory–accuracy trend of Experiments 2 and 3. At similar physical sizes, frozen SnapKV scores higher than StreamingLLM on VT, while StreamingLLM is stronger on KV and PS at every budget. Because the methods preserve different context representations, the ranking suggests that effective retention depends on the evidence required by each task, consistent with Corollary B.2's focus on distinguishable future answers.

Table 5: Experiment 4 complete allocated state, in MiB (2^{20} bytes). KV, Pos., and Quant. give answer-time payload, position buffers, and quantization metadata. Other includes BF16 residuals, logits, and controller state. Answer and Reuse are the complete answer-time and reusable-prefix allocations; A and R give percentages of their matched Full-KV states. Entries are medians over 90 contexts after taking the median over five requests per context, so component medians need not sum to the median total. Public weights and temporary construction memory are excluded.

Method	ρ	KV	Pos.	Quant.	Other	Answer	Reuse	A (%)	R (%)
Full KV	–	16701.5	32.6	0.0	0.250	16734.4	16644.6	100.0	100.0
StreamingLLM	1/8	2076.5	4.1	0.0	0.250	2080.8	2080.6	12.4	12.5
	1/4	4153.0	8.1	0.0	0.250	4161.4	4161.2	24.9	25.0
	1/2	8306.1	16.2	0.0	0.250	8322.6	8322.3	49.8	50.0
	3/4	12459.1	24.3	0.0	0.250	12483.7	12483.5	74.6	75.0
SnapKV (frozen)	1/8	2165.9	33.8	0.0	0.250	2200.0	2109.0	13.1	12.7
	1/4	4242.4	66.3	0.0	0.250	4309.0	4217.9	25.7	25.3
	1/2	8395.5	131.2	0.0	0.250	8526.9	8435.9	50.9	50.7
	3/4	12548.5	196.1	0.0	0.250	12744.8	12653.8	76.1	76.0
SnapKV (query-aware)	1/8	2079.2	32.5	0.0	0.250	2112.0	16644.6	12.6	100.0
	1/4	4158.4	65.0	0.0	0.250	4223.7	16644.6	25.2	100.0
	1/2	8316.8	129.9	0.0	0.250	8447.0	16644.6	50.5	100.0
	3/4	12475.1	194.9	0.0	0.250	12670.3	16644.6	75.7	100.0
KIVI 4-bit	–	4157.4	0.0	1039.4	3.604	5200.7	5193.7	31.1	31.2
KIVI 2-bit	–	2078.7	0.0	1039.4	3.604	3122.0	3117.7	18.7	18.7

KIVI at 4 bits uses 31.1% of the Full-KV answer-time physical size while preserving 96.0% of Full-KV-correct queries on average across tasks (Preserve). It matches Full KV's KV accuracy, with PS and VT scores also close to the reference. Reducing precision to 2 bits lowers size to 18.7% and Preserve to 37.1%, with larger losses on KV and PS than on VT. Both settings retain every position, so the contrast illustrates why Remark B.14 counts precision and additional persistent state alongside positions. As with grouped codebook in Experiment 2, the representation determines how effectively a smaller cache supports retrieval.

Query-aware SnapKV improves task scores by 0.13–71.33 percentage points across the 12 comparisons, with confidence intervals entirely above zero in 11. For example, at $\rho = 1/2$ on KV Retrieval, the gain is 66.67 percentage points with a 95% interval of $[56.00, 76.67]$. The remaining comparison is Variable Tracking at $\rho = 3/4$, where the gain is 0.13 points with an interval of $[-0.67, 0.93]$. The positive gains across tasks extend Experiment 3's observed timing advantage to SCBench. Query-aware SnapKV leads the position-selection methods on KV and VT at every budget, although StreamingLLM is stronger on PS at the two smallest budgets. The gains agree qualitatively with part (b) of Theorem 2.1, because knowing the request allows selection to prioritize information for its answer. Here, query processing and first-token logits use Full KV before selection, so the advantage concerns the post-query protocol.

Table 5 shows that KIVI reduces both answer-time and reusable allocation after quantization metadata and BF16 residuals are counted. Because both precisions retain every position, their different sizes illustrate the precision dependence in Remark B.14. Query-aware SnapKV instead retains 100% reusable state at every budget because each request selects from Full KV. Its smaller answer-time allocation therefore describes the state used for one request. The distinction agrees with Corollary B.12's requirement to count the complete retained source, although the SCBench measurements do not establish the source and error assumptions of the recall bounds.

C.4.3 Part B: Causal cache-use settings

Part B tests whether replacing the queried record's retained KV entries moves the generated answer toward the value represented by a donor context. The intervention complements Part A because a behavioral advantage or literal record retention alone does not establish that those entries influence the answer.

Paired benchmark and bidirectional transfer. We reuse all five queries from each of Part A's 30 KV Retrieval sessions, yielding 150 pairs in 30 context clusters. Each pair changes only the queried value to a distinct lowercase UUIDv4 string generated with seed 47, excluding collisions

with existing keys, values, or answers. Tokenization preserves input length and all tokens outside the edited value and its boundary. Keys, other records, queries, and prompts remain unchanged. Pairs and source–target labels are fixed before evaluation without correctness filtering, with the original context permitted on either side.

For each pair and cache condition, we evaluate both natural caches and both directions of queried-record KV transfer, yielding four answers. The original-context answer is reused from Part A after matching inputs, model, cache condition, and decoding. The other three answers are generated in Part B. Transfers replace keys and values across all layers and KV heads while preserving recipient position maps and state outside the transfer region. Full KV transfers the aligned record span. SnapKV transfers only positions retained in both caches, with empty intersections marked unavailable and missing positions left absent. KIVI expands the transfer to complete quantization groups, including packed codes, scales, minima, and residuals. The effective span and bytes are recorded because neighboring positions may be included.

Model and cache conditions. Part B uses Part A's frozen BF16 model, prompts, greedy decoding, and 512-token cap. It evaluates 11 cache conditions: Full KV, frozen and query-aware SnapKV at the same four retention fractions, and KIVI at 4 and 2 bits. StreamingLLM remains confined to Part A. The resulting grid contains $150 \times 11 = 1{,}650$ pair–condition comparisons. Seven conditions have a pre-query intervention boundary, while the four query-aware SnapKV conditions have a post-query boundary.

Answer categories and causal effects. We score each extracted answer against both reference values using Part A's KV scorer, classifying it as source-only, target-only, both, or neither. An answer containing both values receives credit for both references, but does not count as an exclusive switch. To quantify the effect of transfer in each direction, we measure changes in donor-value and recipient-value scores relative to the unmodified recipient. The exclusive donor-switch rate is the fraction of initially recipient-only answers that become donor-only after transfer. Unavailable transfers and missing results are excluded from the corresponding effect denominator and recorded separately.

C.4.4 Part B: Causal cache-use results

Table 6 shows that Full-KV record transfer redirects answers toward the donor value, with 94.7–95.3% donor accuracy across the two directions, matching the donors' natural accuracies. Because the query and cache outside the transferred record remain fixed, the redirected answers establish causal usefulness of the record KV, complementing the behavioral evidence in Part A and Experiments 2 and 3. Recipient accuracy is zero throughout the table, so all donor-correct answers are donor-only. The paired contexts require different answers to the same query and thus exemplify future distinguishability in Corollary B.2. Redirecting the answer through persistent state also illustrates Theorem 2.1's interpretation of the cache as a message to subsequent computation.

The effect survives 4-bit KIVI, whose transfers of complete quantization groups achieve 96.0–96.7% donor accuracy. The compressed representation therefore retains causally useful value information, complementing its high retrieval performance in Part A. At 2 bits, donor accuracy falls to 25.3–26.0%, paralleling the decline in natural retrieval. Since both precisions retain every position, the contrast reinforces the importance of precision for both ordinary retrieval and causal transfer.

For both SnapKV variants and directions, natural and donor accuracies increase monotonically with ρ, although all transfers remain partial, replacing only record positions retained in both caches. Query-aware SnapKV yields higher donor accuracy at the three smaller budgets, while frozen SnapKV is higher at $\rho = 3/4$. At that budget, query-aware donor accuracy is 37.3–42.0%, below its 95.3–96.0% natural accuracy. The gap is consistent with compatibility between transferred entries and the remaining state affecting retrieval, because pre-query transfer recomputes the query, whereas post-query transfer preserves the recipient's query KV and first-token logits. Remaining errors may reflect information loss or difficulty using retained information. The timing dependence therefore motivates the complete-state perspective of Theorem 2.1, which includes retained records and query-conditioned continuation state.

Table 6: Answer accuracy before and after bidirectional record transfer in Experiment 4, Part B, on SCBench KV Retrieval with frozen BF16 Ministral 3 8B Instruct 2512. All accuracy values and their errors are expressed as percentages. Before transfer measures accuracy against the recipient's value before intervention. Donor and Recipient measure accuracy against the respective values after transfer. S and T denote the paired contexts, with arrows pointing from donor to recipient. Budget gives the token-retention fraction ρ or KIVI code precision. Each condition uses 150 original and counterfactual pairs from 30 contexts, with five queries per context and edits sampled under seed 47. Scores are averaged within each context and then across contexts. Entries show mean official score (%) $\pm$ half the width of the 95% percentile confidence interval from $10{,}000$ paired bootstrap resamples of whole context clusters within each task, using bootstrap seed 172.

Method	Budget	Transfer	Before transfer	Donor	Recipient
Pre-query transfer					
Full KV	Full	$S \leftarrow T$	95.3 ± 2.7	94.7 ± 3.7	0.0
		$T \leftarrow S$	94.7 ± 3.7	95.3 ± 2.7	0.0
SnapKV	$3/4$	$S \leftarrow T$	70.0 ± 7.7	63.3 ± 9.3	0.0
		$T \leftarrow S$	68.7 ± 8.3	62.7 ± 8.7	0.0
	$1/2$	$S \leftarrow T$	30.7 ± 10.3	22.7 ± 9.0	0.0
		$T \leftarrow S$	27.3 ± 9.0	25.3 ± 10.7	0.0
	$1/4$	$S \leftarrow T$	7.3 ± 5.0	3.3 ± 3.7	0.0
		$T \leftarrow S$	7.3 ± 6.7	6.0 ± 4.7	0.0
	$1/8$	$S \leftarrow T$	1.3 ± 1.7	1.3 ± 1.7	0.0
		$T \leftarrow S$	2.0 ± 2.0	1.3 ± 1.7	0.0
KIVI	4-bit	$S \leftarrow T$	96.0 ± 3.0	96.0 ± 3.0	0.0
		$T \leftarrow S$	96.0 ± 3.0	96.7 ± 2.7	0.0
	2-bit	$S \leftarrow T$	22.0 ± 7.3	26.0 ± 7.7	0.0
		$T \leftarrow S$	20.7 ± 8.0	25.3 ± 8.3	0.0
Post-query transfer with query-aware SnapKV					
SnapKV	$3/4$	$S \leftarrow T$	95.3 ± 2.7	42.0 ± 6.7	0.0
		$T \leftarrow S$	96.0 ± 2.7	37.3 ± 6.7	0.0
	$1/2$	$S \leftarrow T$	93.3 ± 3.3	39.3 ± 6.3	0.0
		$T \leftarrow S$	92.0 ± 4.3	33.3 ± 6.7	0.0
	$1/4$	$S \leftarrow T$	68.0 ± 8.3	22.7 ± 6.7	0.0
		$T \leftarrow S$	75.3 ± 7.7	20.0 ± 6.7	0.0
	$1/8$	$S \leftarrow T$	32.7 ± 7.7	6.0 ± 3.7	0.0
		$T \leftarrow S$	29.3 ± 10.0	3.3 ± 3.0	0.0