CRYSTALCACHE: CROSS-DOMAIN TRANSFER FROM COGNITIVE MEMORY CRYSTALLIZATION TO KV CACHE EVICTION IN LONG-CONTEXT LLMS

Po-Ting Lin
Independent Researcher
Taiwan (ROC)
botimlin@gmail.com

ABSTRACT

The Key–Value (KV) cache of long-context Large Language Models (LLMs) grows linearly with context length and is now the dominant memory bottleneck of long-context inference; at 128K tokens a single batch of bf16 KV for Llama-3-8B already exceeds the model weights themselves. Existing eviction methods fall into two generations. The first generation (H2O, SnapKV, StreamingLLM, Scissorhands) summarises each token by a single scalar and evicts at token granularity, producing "coverage holes" over semantically coherent passages. The second generation (ChunkKV, EpiCache, CAOTE, DefensiveKV, PyramidKV) advances along a single axis each—fixed-size grouping, signal fusion, or robust aggregation of repeated observations—but *none* simultaneously satisfies the four structural requirements of dynamic semantic boundaries, two independent scoring dimensions, an explicit rarity signal, and progressive (rather than binary) retention.

We propose **CrystalCache**, a KV-cache eviction algorithm derived from the structural predictions of the *Crystallization Memory Framework*: that any system serving a memory function should describe each item along at least two independent axes (analogous to a crystal's structural extent and formation strength) and should organise items as a multi-branch trunk rather than a single block. CrystalCache instantiates these predictions in four concurrent design moves: (1) it builds *trunks*—semantic units bounded by sentence punctuation and refined by co-attention—rather than fixed-size chunks or utterance clusters; (2) it scores each trunk along two independently computed dimensions, an associative crystallization term D (structural centrality in the trunk graph) and an encoding impact term M_i (attention salience plus a Von Restorff rarity term), and composes them as $\text{score} = \max\big(D,\, \alpha \cdot \text{normalize}(\log(1 + M_i))\big)$, providing two independent survival paths; (3) it injects an *explicit* token-frequency rarity signal $U_i = 1/(1 + \log(1 + c_i))$ directly into the score, a signal absent from all four contemporaneous works; and (4) it replaces binary retention with a two-stage *branch dissolution* procedure that performs proportional retention between trunks and Mi-ranked retention within trunks.

On Llama-3.1-8B-Instruct, Mistral-7B-Instruct-v0.3, and Qwen3-8B, across Needle-in-a-Haystack and a Delayed Association diagnostic at retention budgets $\beta \in \{0.3, 0.5\}$, CrystalCache wins all $3 \times 2 \times 2 = 12$ retrieval comparisons against H2O, SnapKV, ChunkKV, StreamingLLM, and PyramidKV; on Qwen3-8B Needle ($\beta = 0.5$) it doubles the best baseline (0.333 vs. 0.167) and quadruples the weakest (vs. 0.083). Ablations identify the Von Restorff rarity term as the single most impactful component (-0.383 when removed), confirm that trunk-level eviction outperforms token-level (-0.317 when $T_{\max} = 1$), and confirm that the dual-dimension max composition strictly beats either dimension alone. On the broader-coverage LongBench suite, CrystalCache is competitive but not leading, a trade-off we attribute to the spatial-coverage cost of trunk-level retention and discuss honestly as a limitation. The end-to-end system delivers 50–70% steady-state decode memory savings; the prefill overhead (54–64% at 16K–32K) stems entirely from a CPU-NumPy $O(n^2)$ co-attention edge extraction and is engineering, not algorithmic.

Beyond the empirical result, the consistency of the 12/12 cross-model, cross-task, cross-budget gains constitutes a computational corroboration of the structural predictions of the Crystallization Memory Framework: when a system serves a memory function, structural principles derived from biological memory transfer non-trivially to its design.

***K*eywords** KV cache eviction · long-context inference · large language models · memory crystallization · cross-domain transfer · cognitive memory

1 Introduction

1.1 The KV Cache Bottleneck

Autoregressive decoding in modern Large Language Models (LLMs) caches the key and value projections of every previously seen token so that the attention layer at step $t+1$ does not need to recompute them. The resulting Key–Value (KV) cache grows linearly with context length: for a model with L layers, H_{kv} KV heads, and head dimension d, the cache for a single sequence of length n at b bytes per element occupies

$$\text{Mem}_{\text{KV}} \;=\; 2\,L\,H_{\text{kv}}\,d\,n\,b. \tag{1}$$

bytes. For Llama-3.1-8B ($L = 32$, $H_{\text{kv}} = 8$, $d = 128$) at $n = 128$K in bf16, this is roughly 16 GB *per sequence*—larger than the model weights themselves. The cache, not the weights and not compute, is the binding constraint of long-context inference, and is the reason long-context serving on commodity hardware so often falls back to small batch sizes, aggressive context truncation, or refusal of long requests outright.

A natural response is *eviction*: at some point during prefill (or periodically during decode), discard the KV entries of tokens deemed least likely to be attended to in future steps, and keep only a budget of $B \ll n$ entries. The question is which $n - B$ tokens to drop. Two generations of methods have proposed answers.

1.2 Two Generations of KV Eviction, and What They Share

First generation: single scalar, single token. H2O [1], SnapKV [2], StreamingLLM [3], and Scissorhands [4] differ in their importance signals—accumulated past attention, look-ahead window attention, attention sinks plus a sliding window—but share two implicit assumptions: **(A1)** a token's importance is adequately summarised by a single scalar, typically derived from attention; and **(A2)** the eviction unit is the individual token, with no notion of semantic grouping. These assumptions interact poorly with retrieval-style queries. Consider Needle-in-a-Haystack [5]: a single critical fact is written once, deep inside the context, far from the eventual query. Its accumulated attention is small, its temporal proximity to the query is poor, and so as the budget tightens it is among the first to be discarded—a *coverage hole* carved out of an otherwise dense semantic region.

Second generation: partial relaxations. Four roughly contemporaneous works (2025) each loosen one of the two assumptions, but no single one loosens both at the joint where it matters. ChunkKV [6] replaces the token unit with fixed-size chunks of 10 consecutive tokens, ranked by intra-chunk attention sum; the unit is grouped, but the boundary is purely positional and the score remains a single scalar. EpiCache [7] clusters utterances via sentence-embedding K-Means into "episodes" and ranks episodes by a patched-prompt cross-attention scalar; the granularity is much coarser than a single token but is tied to conversational structure rather than to general document structure. CAOTE [8] fuses the attention score α_j with the value vector v_j into a closed-form output-error estimator $\alpha_j/(1-\alpha_j) \cdot \|VA^\top - v_j\|_2$; the score is multi-source, but it is still a single scalar at the individual-token level. DefensiveKV [9] observes that averaging repeated attention observations erases rare-but-critical peaks and replaces the average with a max plus a prior risk correction; the aggregator becomes robust, but the unit and the dimensionality of the score do not change.

The structural gap. Across both generations no method simultaneously satisfies the conjunction of *dynamic semantic boundaries*, *two independently computed scoring dimensions*, an *explicit token-frequency rarity signal*, and *progressive (rather than binary) retention*. Each contemporaneous work moves along one axis; none moves along all four. The empirical consequence, as we will document, is that on long-distance retrieval tasks all of them leave a sizeable gap to the full-cache upper bound even at $\beta = 0.5$, and that the gap is not closed by stacking several of them naively.

1.3 From Memory Crystallization to KV Eviction

The structural shape of this gap—"one number per item, all-or-nothing decisions, no built-in notion of rarity"—has a striking resonance with an old debate in the cognitive memory literature. Single-scalar models of memory strength

such as Rescorla–Wagner [10] struggle in the same way: they cannot simultaneously represent "a large but loosely consolidated trace" and "a small but extremely sharply encoded trace," because both are compressed into one number. The Crystallization Memory Framework [11] addresses precisely this representational bottleneck. It models a memory trace as a growing crystal characterised by *at least two independent physical properties*—structural extent (how much has formed) and formation strength (how harshly it was forged, which determines its resistance to dissolution)—and it organises traces as multi-branch trunks rather than monolithic blocks, so that each branch represents one feature and the trunk's overall consolidation depends on the joint state of its branches.

Two structural predictions of the framework are independent of any specific cognitive substrate: **(P1)** a system that serves a memory function should describe each item along at least two independent dimensions, not one; **(P2)** a system that serves a memory function should organise items as feature-level branches under a higher-level grouping, not as flat lists. The KV cache of a long-context LLM is, functionally, an enormous working memory: it is the substrate the model interrogates when answering a query about distant context. The central question of this paper is whether (P1) and (P2)—predictions originally framed in biological terms—transfer non-trivially to this artificial memory.

Concretely, we ask: *If we rebuild a KV-cache eviction algorithm so that it (i) describes each retention candidate along two independent dimensions and (ii) operates over a trunk-and-branch organisation, do we obtain a measurable, reproducible, cross-model improvement on the tasks where the structural gap above is most pronounced?*

1.4 Contributions

We instantiate (P1) and (P2) as a concrete eviction algorithm, **CrystalCache**, and evaluate it on three open-weight LLMs across two long-distance retrieval tasks at two retention budgets. Our contributions are:

C1. Dynamic semantic trunks as eviction units. We construct *trunks* via a three-step procedure—sentence-boundary segmentation, co-attention-driven merging, and a maximum-size split—so that the boundary of each eviction unit is decided jointly by linguistic structure and by the model's own attention dynamics. This is distinct from ChunkKV's purely positional fixed-10-token chunks [6] and from EpiCache's utterance-level K-Means clusters [7].

C2. Two-dimensional score with two independent survival paths. We replace the single-scalar score with

$$\text{score}(g) \;=\; \max\big(D(g),\; \alpha \cdot \text{normalize}(\log(1 + \overline{M_i}(g)))\big), \tag{2}$$

where D measures structural centrality in the trunk graph and M_i measures encoding impact. Crucially the two dimensions are computed from *independent* signals (graph connectivity vs. token-level salience and rarity), and they are composed by $\max$ rather than weighted average so that a trunk excelling along either dimension survives. This is a strictly different operation from DefensiveKV's $\max$ over multiple noisy observations of the *same* attention signal [9] and from CAOTE's closed-form fusion of attention and value into a *single* scalar [8].

C3. Explicit Von Restorff rarity signal at the token level. We embed token frequency directly into the score as $U_i = 1/(1 + \log(1 + c_i))$, instantiating the Von Restorff distinctiveness effect [12] as an *attention-independent* input to eviction. To our knowledge none of the four contemporaneous second-generation works includes such a signal: ChunkKV, EpiCache, and CAOTE do not; DefensiveKV's robustness operates at the *attention observation* layer, not at the token-frequency layer, and is therefore mechanistically orthogonal to ours. Our ablation (Section 6) shows this single component to be the largest contributor, with removal causing a -0.383 absolute drop on Llama-3.1-8B Needle at $\beta = 0.5$.

C4. Two-stage branch dissolution in place of binary retention. We replace the all-or-nothing retain/discard decision with a two-stage *branch dissolution* mechanism: a first stage allocates retention proportions across trunks bottom-up (low-scoring trunks lose proportionally more tokens, with a minimum-survival threshold $N_{\min}$), and a second stage selects which specific tokens to keep within each partially-retained trunk by token-level M_i ranking. To our knowledge, two-stage progressive retention with both an inter-unit and an intra-unit dimension is novel in the KV-eviction literature.

C5. Cross-model, cross-task, cross-budget empirical consistency. On Llama-3.1-8B-Instruct, Mistral-7B-Instruct-v0.3, and Qwen3-8B—three models with different training data, attention architectures (MHA, GQA), and positional encodings—CrystalCache wins all $3 \times 2 \times 2 = 12$ comparisons against H2O, SnapKV, ChunkKV, StreamingLLM, and PyramidKV on Needle-in-a-Haystack and Delayed Association at $\beta \in \{0.3, 0.5\}$. The Qwen3-8B Needle result at $\beta = 0.5$ doubles the best baseline (SnapKV, $0.167 \to 0.333$) and quadruples the weakest (ChunkKV, $0.083 \to 0.333$). The consistency of these gains across heterogeneous models constitutes the empirical content of our cross-domain transfer claim: the structural predictions of the Crystallization Memory Framework do not merely fit one model and one task; they generalise.

1.5 Honest Scope

We additionally evaluate on LongBench [13], where CrystalCache is competitive but not leading—a result we attribute to the spatial-coverage cost of trunk-level retention on broad-coverage tasks (summarisation, multi-document QA) where every region of the context contributes a little. We report this honestly in Section 5 rather than omitting it. Similarly, our prefill overhead at 32K is 64%, dominated entirely by a CPU-NumPy $O(n^2)$ co-attention edge extraction that has obvious GPU and locality-sensitive-hashing optimisations not yet implemented; the algorithmic cost of trunk construction, graph computation, and dissolution combined is under 170 ms at all tested context lengths. Finally, our decode-time mechanism (a 64-step "breath cycle" of D-decay, resonance-based revival, Hebbian edge evolution, and re-eviction; Section 3.8) is implemented but not exercised in the experiments reported here, since our maximum generation length of 50 tokens is shorter than the breath interval. We mark this clearly throughout.

1.6 Paper Organisation

Section 2 reviews KV-cache eviction and the cognitive memory background and positions CrystalCache against contemporaneous work on four structural axes. Section 3 presents the full algorithm: prefill pipeline, trunk construction, M_i and D computation, score composition, branch dissolution, and the (currently dormant) decode-time mechanism. Section 4 describes models, tasks, baselines, and evaluation protocol. Section 5 reports main results on Needle, Delayed Association, and LongBench. Section 6 ablates each component. Section 7 decomposes the computational cost. Section 8 discusses limitations and the cognitive-theoretic interpretation. Appendices document hyperparameters, the Delayed Association construction, the per-task LongBench breakdown, and full algorithm pseudocode.

2 Background and Related Work

This section serves three purposes. Section 2.1 fixes the technical setting and the units in which we will discuss memory cost. Sections 2.2 and 2.3 survey the two generations of KV-cache eviction methods, organising them not by chronology but by the structural assumptions they make about the eviction problem. Section 2.4 introduces the cognitive memory background, isolates two substrate-neutral structural predictions of the Crystallization Memory Framework, and connects them to the Von Restorff effect that we will use directly. Section 2.5 positions CrystalCache against this landscape and states precisely the falsifiable content of our cross-domain transfer claim.

2.1 Transformer Attention and the KV Cache

A standard decoder-only Transformer [14] layer maps an input $X \in \mathbb{R}^{n \times d_{\text{model}}}$ to queries, keys, and values via three projections and computes

$$\text{Attn}(Q, K, V) \;=\; \text{softmax}\left(\frac{QK^\top}{\sqrt{d}} + M_{\text{causal}}\right) V, \tag{3}$$

where M_{causal} enforces causality and d is the per-head dimension. During autoregressive decoding the model emits tokens one at a time, and each step's attention layer must attend over the entire history. Recomputing K and V for the full history at every step would scale as $O(n^2 d)$ per step and is infeasible. Production systems therefore cache the per-layer K and V tensors of every previously seen token in a structure conventionally called the *KV cache*, so that each decode step performs only $O(nd)$ work for the new query against a precomputed history.

For a model with L layers, H_{kv} KV heads, head dimension d, sequence length n, batch size B, and b bytes per element, the cache consumes

$$\text{Mem}_{\text{KV}} \;=\; 2\, L\, H_{\text{kv}}\, d\, n\, B\, b \tag{4}$$

bytes; the factor of 2 accounts for K and V separately. Modern architectures aggressively reduce H_{kv} via Grouped-Query Attention (GQA): Llama-3.1-8B uses $H_{\text{kv}} = 8$ KV heads shared across 32 query heads. Even with GQA, a single bf16 sequence at $n = 128$K consumes ~ 16 GB—comparable to the model's ~ 16 GB of weights. Multi-tenant serving must therefore choose between long context, large batch, and many concurrent users, but cannot have all three simultaneously. The KV cache, not the weights and not the compute, is the binding resource of long-context inference.

The natural response is *eviction*: at some point during prefill (or periodically during decode), discard the KV entries of tokens deemed least likely to be attended to in future steps, and keep only a budget $B \ll n$. The eviction problem decomposes into three sub-decisions:

- **Unit:** What is the smallest entity over which eviction operates—a token, a fixed-size group, a semantic group?
- **Score:** How is each unit's retention priority computed—from one signal or several, in one dimension or several?

- **Decision rule:** Given scores, how is the budget enforced—all-or-nothing per unit, or with progressive partial retention?

We will use this three-axis decomposition to organise the existing literature in the next two subsections, and again in Table 1 when we position CrystalCache.

2.2 First-Generation Eviction: One Scalar Per Token

The first wave of KV-eviction methods can be characterised by two implicit assumptions:

- **(A1)** A token's retention priority is adequately summarised by a single scalar.
- **(A2)** The eviction unit is the individual token; there is no semantic grouping.

The methods differ in *which* scalar they choose and in how it is observed, but they share these two assumptions and consequently share a common failure mode on long-distance retrieval.

H2O [1]. The Heavy-Hitter Oracle hypothesis: tokens that have accumulated high attention so far will continue to attract disproportionate attention in the future. The score for token i at decode step t is the running sum

$$\mathrm{score}_i^{\mathrm{H2O}}(t) \;=\; \sum_{q \le t} A_{q,i}, \tag{5}$$

and the cache is pruned to the top-B heavy hitters plus a most-recent window. The assumption is empirically defensible when the relevant content has already been attended to during context, but breaks down when the model has not yet had a reason to attend to a token—the canonical Needle case, where the relevant fact has been encoded but never queried.

SnapKV [2]. Uses the final w tokens of the prefill (the "observation window") as a query-side proxy. For each KV head independently, it extracts the attention pattern from the observation window over the full preceding context, applies a smoothing pool, and retains the top-B positions per head. This is a one-shot prefill-time decision; no further eviction occurs during decode. Query awareness improves over H2O when the prompt's tail is informative about the answer, but degrades when the relevant context is far from the tail or when the question itself is short and uninformative about which content matters.

StreamingLLM [3]. The first few tokens act as "attention sinks"—softmax mass concentrates on them regardless of their semantic content. StreamingLLM proposes preserving the first N_{sink} tokens together with a sliding window of the most recent W_{recent} tokens and discarding everything in between. This is essentially a strong inductive bias rather than a learned ranking: it accepts that all middle tokens will be lost in exchange for stability of the attention distribution, and serves as an upper-budget bound on what any sink-plus-window scheme can do.

Scissorhands [4]. A parallel formulation of the heavy-hitter intuition with an explicit "persistence of importance hypothesis": tokens that received high attention in a sliding observation window are likely to continue receiving high attention. Mechanically this is an H2O variant scored over a windowed rather than cumulative observation horizon; it shares (A1) and (A2).

PyramidKV [15]. PyramidKV adds an orthogonal axis that the others lack: a per-layer budget allocation, with more KV preserved in early layers and less in late layers, motivated by the observation that early layers exhibit broader attention while late layers attend to fewer positions. The underlying scoring is still single-scalar per token, but the budget is no longer uniform across layers. We list it here because its scoring axis is first-generation; the layer-wise allocation is orthogonal to anything we propose and is in principle composable with CrystalCache.

The shared failure mode. All five methods reduce the per-token observation (cumulative attention, look-ahead window attention, fixed positional priors, or windowed observations) to one scalar and decide retention per token independently. There is no notion of *semantic group*—two adjacent tokens forming a single fact ("Tom Hanks") can be split apart at the eviction boundary—and no notion of a second, attention-independent dimension that would allow a token to survive for reasons other than "it has been or will be attended to." On Needle-in-a-Haystack the consequence is direct: the Needle has been written once, far from the query, and is unlikely to dominate the cumulative or look-ahead attention statistics that drive the score. It is therefore among the first tokens evicted, producing what we call a *coverage hole* in the otherwise dense neighbourhood that contains it.

2.3 Second-Generation Eviction: Partial Relaxations

Four roughly contemporaneous works (all 2025) each loosen one of the assumptions A1 or A2, but no single one loosens both at the joint we will identify in Section 2.5. We describe each in turn and then summarise.

ChunkKV [6]: positional grouping. Replaces the per-token unit with fixed-size chunks of 10 consecutive tokens. The per-chunk score is the sum of intra-chunk attention, and eviction operates at chunk granularity:

$$\text{score}^{\text{ChunkKV}}(c) \;=\; \sum_{i \in c} \sum_{q} A_{q,i}, \tag{6}$$

where c ranges over fixed 10-token chunks. This relaxes A2 (the unit is now a group), but the boundary is purely positional—it bears no relation to whether the chunk is a coherent semantic unit—and the score is still a single scalar.

EpiCache [7]: utterance-level clustering for conversational data. Designed for long conversational QA. Performs sentence-embedding K-Means clustering at *utterance* granularity (using an external embedding model) to form "episodes," then ranks episodes by cross-attention from a patched query prompt. The unit is now meaningfully semantic but very coarse—one cluster per utterance is much larger than a sentence—and the construction is structurally tied to multi-turn conversational data with explicit turn boundaries. A document without conversational structure cannot be clustered in this way without either fabricating utterance boundaries or falling back to a sentence-level approximation that the method does not specify.

CAOTE [8]: signal fusion via attention-output error. Argues that pure attention-based scoring is a poor proxy for what actually matters—the resulting attention output error—and derives a closed-form per-token score combining the attention coefficient α_j and the value vector v_j:

$$\text{score}_j^{\text{CAOTE}} \;=\; \frac{\alpha_j}{1-\alpha_j} \cdot \left\| V A^\top - v_j \right\|_2. \tag{7}$$

This relaxes A1 in the sense that the score is now multi-source (attention and value), but it is still mathematically a single scalar per token, and it still operates per token. The two ingredients are fused at the score level into one number; there is no notion of two independent paths that a token could take to survive eviction.

DefensiveKV [9]: robust aggregation of repeated observations. Targets a specific failure mode of A1-style scoring: when one aggregates multiple noisy attention observations of the same token by averaging, rare-but-critical attention peaks are washed out. DefensiveKV replaces the average with a max plus a prior risk correction:

$$\text{score}_i^{\text{Def}} \;=\; \max_{q \in \mathcal{O}_i} A_{q,i} \;-\; \rho\, \text{Risk}_i, \tag{8}$$

where $\mathcal{O}_i$ is the set of observation queries for token i and ρ is a prior risk weight. This is a meaningful robustification of how a noisy signal is aggregated, but it operates entirely at the level of *repeated observations of the same attention signal*—it does not introduce a second, mechanistically independent dimension, and it does not change the per-token unit. In the framework of Section 2.1 it modifies the score along the aggregator axis but leaves the unit, the dimensionality, and the decision rule unchanged.

What the second generation does and does not change. Each method moves along a different axis: ChunkKV along the unit-granularity axis (token $\to$ fixed chunk), EpiCache along the unit axis but only for conversational data (token $\to$ utterance), CAOTE along the signal-source axis (attention only $\to$ attention + value, fused), and DefensiveKV along the aggregator axis (mean $\to$ robust max). What none of them changes is the dimensionality of the score (still one) and the retention decision (still binary per unit). And what none of them introduces is a token-frequency rarity signal: whether a token has appeared once or one hundred times in the context plays no role in any of the four scores above. We will return to this gap in Section 2.5.

2.4 Cognitive Memory Models and the Crystallization Framework

Quantitative models of biological memory have, for fifty years, oscillated between two camps. Single-scalar models—of which Rescorla–Wagner [10] is the prototypical example—summarise the strength of an associative trace by one number V that is updated by an error signal δ each time the cue is encountered:

$$V_{t+1} \;=\; V_t + \alpha\beta\, \delta_t, \qquad \delta_t \;=\; \lambda - V_t, \tag{9}$$

where λ is the asymptotic strength, α is a cue-salience parameter, and β is a reinforcement-effectiveness parameter. The model is mathematically clean, computationally cheap, and fits a wide variety of acquisition curves. It is also famously inadequate for several phenomena: it cannot simultaneously represent a trace that is large but easily extinguished and a trace that is small but extremely persistent, because both must compress into the single number V. It also fails to predict effects in which the structure of the encoding event (its surprise, its distinctiveness, its emotional intensity) modulates persistence *independently* of cumulative repetition.

The natural response is to admit a second dimension. Multiple-trace and multiple-system models do exactly this in various forms. What is less common, and what we will rely on, is a unified framework that derives *structural predictions* about what such a second dimension must look like, regardless of substrate.

The Crystallization Memory Framework. **(author?)** [11] models a memory trace by analogy to a growing crystal. Two structural claims are central.

Claim 1: a memory trace has at least two independent properties. A crystal has both

- a *structural extent* D—how much has formed, by accumulated growth events; and
- a *formation strength* M—the harshness of the conditions under which it was forged, which determines its resistance to dissolution.

These two properties are independent: a crystal can be "large but loosely formed and easily dissolved" (high D, low M) or "small but forged under extreme conditions and unusually durable" (low D, high M). No single number can describe both classes simultaneously, because their survival mechanisms are different. The Rescorla–Wagner-style single-scalar models of memory correspond exactly to compressing this two-property description into one.

Claim 2: memories are organised as multi-branch trunks, not monolithic blocks. A trace is not a single quantity but a tree-structured object: a trunk with multiple branches, each branch corresponding to one perceptual, semantic, or contextual feature. The trunk's overall consolidation depends on the joint state of its branches: branches that fall below threshold are pruned (*branch dissolution*), and the trunk's persistence is gated by how many branches still contribute. Crucially, the dissolution is *progressive at the branch level*, not binary at the trunk level: the trunk degrades one branch at a time rather than vanishing all at once.

Two substrate-neutral structural predictions. The framework's two claims, stated in substrate-neutral form, become two predictions about any system that performs a memory function:

- **(P1)** The system should describe each item along *at least two independent dimensions*, not one, and the two dimensions should provide *independent survival paths* (an item excelling along either dimension should be retained).
- **(P2)** The system should organise items as a *group-level structure with within-group branches*, not as a flat per-item list, and degradation should occur *progressively at the branch level*, not as a binary all-or-nothing decision at the item level.

These predictions are not claims about biology specifically. They are claims about what representational structures any memory system needs in order to remain robust under pressure. If they are correct, they should transfer to artificial systems that perform a memory function.

The Von Restorff effect as a concrete cognitive primitive. Beyond the framework's high-level structural claims, we will use one specific cognitive finding directly as a computational ingredient. **(author?)** [12] demonstrated that an item distinct from its surroundings (the proverbial "red word among black words") is recalled disproportionately well, an effect now known as the Von Restorff or isolation effect. Its mechanism is independent of attention or rehearsal in the usual sense: it is the *rarity of the item against its background* that drives the memory advantage. The simplest mathematical form that captures the rarity-driven advantage with diminishing returns—bounded above, monotonically decreasing in frequency, slow log-scale decay—is

$$\mathrm{Rarity}(c) \;=\; \frac{1}{1+\log(1+c)}, \tag{10}$$

where c is the item's frequency in the surrounding context. We will use exactly Equation (10) in Section 3 as one of the two ingredients of M_i. In a long context for an LLM, the analogue of the original recall experiment is direct: a token that appears once in 32K tokens is, by its very rarity, a strong candidate for retention, while a token that appears in nearly every sentence carries little discriminative value.

Table 1: Positioning of CrystalCache against representative first- and second-generation eviction methods. "Single scalar (fused)" for CAOTE indicates that attention and value are fused into one closed-form scalar; the score is multi-source but one-dimensional. "Robust agg." for DefensiveKV indicates a max aggregator over multiple observations of the *same* attention signal, not a second independent dimension. PyramidKV is omitted from the table because it is orthogonal: it modifies the per-layer budget allocation rather than the per-unit score.

Axis	H2O / SnapKV	ChunkKV	EpiCache	CAOTE / DefensiveKV	**CrystalCache**
Eviction unit	token	fixed chunk (10)	utterance	token	**dynamic semantic trunk**
Boundary criterion	—	positional	sentence-emb. K-Means	—	**sentence + co-attention**
Score dimensions	1 (scalar)	1 (scalar)	1 (scalar)	1 (fused / robust agg.)	**2 (independent)**
Composition	—	—	—	closed-form fusion	$\max$ **over two paths**
Rarity signal	none	none	none	attention-observation level	**token-frequency (Von Restorff)**
Retention granularity	binary	binary	binary	binary	**two-stage progressive**

2.5 Positioning of CrystalCache

The KV cache of a long-context LLM is, functionally, an enormous working memory: the substrate the model interrogates when answering a query about distant context. If P1 and P2 are correct, then a KV-eviction algorithm built around them should outperform algorithms that violate them. Table 1 summarises how CrystalCache and the relevant baselines distribute themselves across the three axes of Section 2.1, augmented with the rarity-signal axis.

Three differentiations that are subtle enough to deserve emphasis. *The two dimensions are independent in a specific, stronger sense.* "Two-dimensional" is occasionally claimed in the eviction literature in the weaker sense of "derived from two ingredients fused into one number" (CAOTE) or "the same number computed under two aggregators" (DefensiveKV). Our D and M_i are independent in the stronger sense: D depends only on the trunk-graph topology, M_i depends only on per-token attention salience and per-token frequency, and the two scores are kept separate up to and including the eviction decision. The composition is $\max$, not a weighted sum, so a trunk that scores high along either dimension survives without any averaging penalty from the other.

The rarity signal lives at the token-frequency layer, not the attention-observation layer. DefensiveKV's robustness operates on multiple noisy observations of the attention signal: the assumption is that some critical tokens have rare-but-large attention peaks that mean-aggregation washes out, and the fix is to use a max aggregator over those observations. Our Von Restorff term operates one layer below: it asks how often the surface token *itself* appears in the context, regardless of any attention statistic. A token that appears once in 32K tokens has $U_i \approx 0.59$; a token that appears 100 times has $U_i \approx 0.18$. This signal exists even before the model produces any attention pattern, and it is therefore mechanistically orthogonal to all four contemporaneous works.

Progressive retention has both an inter-unit and an intra-unit dimension. Token-level binary retention (H2O, SnapKV, CAOTE, DefensiveKV) is the simplest case. Group-level binary retention (ChunkKV, EpiCache) is one step richer but still all-or-nothing per group. Branch dissolution as we will instantiate it in Section 3.7 is two-stage: across trunks we allocate retention proportions bottom-up so that low-scoring trunks lose proportionally more (but not necessarily all) of their tokens, and within each partially-retained trunk we choose which specific tokens to keep by token-level M_i. To our knowledge this combination is novel.

What we will and will not have shown. The cross-domain transfer claim we test is non-trivially falsifiable. If P1 and P2 do not transfer to the LLM KV cache, we should observe (i) that ablating each structural ingredient costs little (the structural design is doing no work and the gain is from elsewhere), and (ii) that the cross-model gains are inconsistent (one model wins, another loses). We will see neither: the ablations show large, ordered drops with the rarity term as the single most impactful component; the cross-model gains are uniformly positive across all 12 retrieval comparisons spanning three architectures, two tasks, and two budgets. This consistency is the empirical content of the claim. We will also see, however, that the gains do not extend uniformly to broad-coverage tasks (LongBench): the trunk-level unit that helps on retrieval has a spatial-coverage cost on summarisation and multi-document QA. Section 5 reports this honestly, and Section 8 interprets it.

Table 2: Notation and defaults. The defaults are those of `configs/default.yaml` and are unchanged across all experiments.

Symbol	Meaning	Default
n	prefill sequence length (tokens)	—
L, H, H_{kv}, d	layers, query heads, KV heads, head dim	model-specific
β	retention budget (fraction of n)	$\{0.3, 0.5\}$
B	target retained tokens, see Eq. (30)	—
$g, \lvert g\rvert$	a trunk and its token count	—
$m \approx n/T_{\max}$	number of trunks after construction	—
E	sparse co-attention edge set $\{(i, j, w_{ij})\}$	—
S_i, U_i, M_i	per-token salience, rarity, encoding impact	—
$D(g), \text{score}(g)$	per-trunk crystallization and composite scores	—
C	chunk size for the eager forward pass	1024
$k_{\text{intra}}, \tau_{\text{intra}}$	intra-chunk top-k and similarity threshold	$8, 0.3$
$k_{\text{cross}}, \tau_{\text{cross}}$	cross-chunk top-k and attention threshold	$4, 0.02$
$T_{\max}$	maximum trunk size	32
τ_{merge}	cross-sentence merge threshold on CAS	0.3
$M_{\min}, M_{\max}$	lower/upper clip on M_i	$0.1, 20.0$
τ_{edge}	trunk-graph edge prune threshold	0.05
s	sigmoid steepness for D	5.0
α	weight of M_i path in score composition	1.0
$N_{\text{sink}}, W_{\text{recent}}$	sink-token count, recent-window size	$4, 128$
$N_{\min}$	minimum tokens a partially-retained trunk keeps	3
T_{breath}	decode-time breath interval (steps)	64

3 Method

3.1 Overview and Notation

We describe CrystalCache as a six-stage prefill pipeline followed by a (currently dormant) decode-time maintenance loop. The prefill pipeline takes as input a sequence of n tokens and a retention budget $\beta \in (0, 1]$, and produces a compressed KV cache of size $B \approx \lceil \beta n \rceil$ that is used by standard autoregressive decoding. All stages described below correspond directly to the production implementation in `crystalcache/`; per-stage entry points are given in Appendix D.

S1. *Chunked eager forward and co-attention edge extraction* (Section 3.2): a chunked prefill in eager attention mode produces, from the first layer's attention tensor, (i) a per-token attention-salience signal and (ii) a sparse co-attention edge set E over tokens.

S2. *Trunk construction* (Section 3.3): tokens are aggregated into semantic units (*trunks*) by sentence-boundary segmentation followed by a cumulative co-attention-driven merge and a maximum-size split.

S3. *Per-token encoding impact* M_i (Section 3.4): the salience signal from S1 is normalised, mixed equally with a Von Restorff rarity term computed from token-frequency counts, and rescaled.

S4. *Per-trunk crystallization score* D (Section 3.5): edges from E are aggregated into a sparse trunk graph; $D(g)$ is a sigmoid of the standardised weighted degree.

S5. *Two-path score composition* (Section 3.6): the per-trunk $\overline{M_i}$ (set in S2 as the top-3 mean of member token M_i) and D are composed by $\max$, with M_i side min–max-normalised over the unprotected trunks at eviction time.

S6. *Branch dissolution* (Section 3.7): a two-stage procedure performs proportional retention between trunks (with an overshoot-on-degenerate-partial rule) and M_i-ranked retention within partially retained trunks. The cache is compressed by index selection.

The decode-time mechanism (Section 3.8) consists of M_i-modulated D decay every step and a 64-step "breath cycle" of activation detection, multi-hop resonance propagation, additive Hebbian edge updates, and re-eviction. We report it in full but note up front that it is *not* exercised by the experiments of Section 5 since our maximum generation length of 50 tokens is shorter than the breath interval; its empirical validation is left to future work.

Notation used throughout is collected in Table 2. The defaults shown are taken from `configs/default.yaml` and are used for all reported experiments.

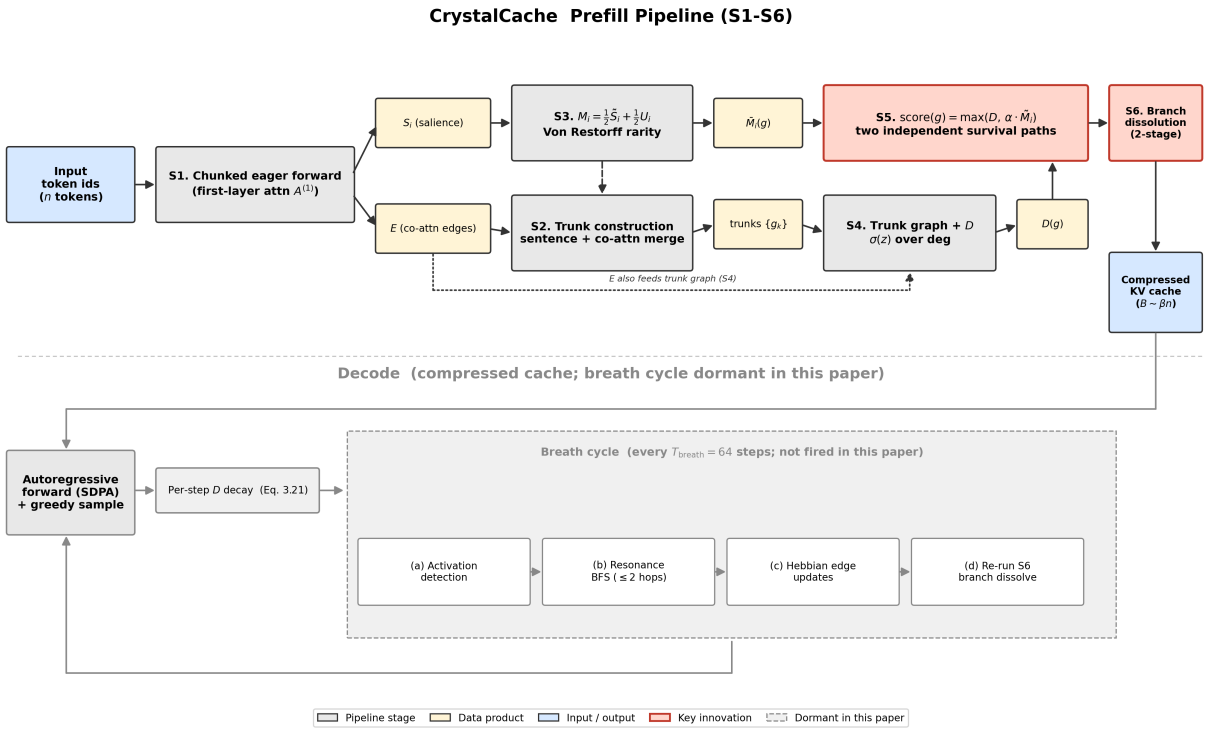

Figure 1: System overview. The prefill stages S1–S6 are executed once; the compressed cache then drives standard autoregressive decoding. The breath cycle (Section 3.8) fires every $T_{\text{breath}} = 64$ decode steps when generation length permits. Solid arrows show data flow; the trunk graph is the bridge between E (S1) and D (S4). Pipeline-stage boxes in light grey, data products in cream, input/output in light blue, key innovations (S5 max composition and S6 branch dissolution) outlined in red. The decode-time mechanism (lower half, dashed grey) is implemented but is dormant in the experiments of Section 5 since our maximum generation length of 50 tokens is shorter than the breath interval.

3.2 Stage S1: Chunked Eager Forward and Co-Attention Edges

Both trunk construction (S2) and the trunk graph used by D (S4) operate on a sparse set of pairwise relations among tokens. We construct this edge set E during a chunked first-layer eager forward pass, taking the union of two sources.

Chunked eager forward. The prefill is performed in chunks of $C = 1024$ tokens. For each chunk we temporarily switch the attention implementation to *eager* so that the first-layer attention tensor $A^{(1)} \in \mathbb{R}^{H \times Q \times K}$ is materialised and exposed, then switch back to SDPA for the remaining layers. Only the first layer's attention is read; later layers run in their default fused implementation. This bounds the eager-attention overhead to one layer and one chunk at a time.

Per-token attention salience (used by S3). For each token at original position i within the chunk, the column corresponding to i as a *key* is summed along the query axis (within the chunk) to yield a per-head per-token salience, after which the top-3 heads are summed:

$$h_h^{(i)} = \sum_{q=1}^{Q} A_{h,q,i}^{(1)}, \qquad S_i = \text{clip}\Big(\sum_{h \in \text{Top3}(h^{(i)})} h_h^{(i)},\ M_{\min},\ M_{\max} \Big), \tag{11}$$

with $M_{\min} = 0.1$ and $M_{\max} = 20.0$. Summing along the query axis (rather than averaging) preserves the total attention mass directed at i; taking the top-3 heads (rather than the per-head mean) reflects the empirical observation that attention is highly head-specific and that uniform-attending heads otherwise dilute the signal by a factor of H.

Intra-chunk edges from attention-pattern cosine similarity. Within a chunk, let $\bar{A}_{\text{chunk}}^{(1)} \in \mathbb{R}^{Q \times K}$ denote the chunk's first-layer attention tensor averaged over heads, restricted to the chunk's own keys. We row-normalise this

matrix:

$$\tilde{A}_{q,\cdot} \;=\; \frac{\bar{A}^{(1)}_{\text{chunk},q,\cdot}}{\big\|\bar{A}^{(1)}_{\text{chunk},q,\cdot}\big\|_2 + \varepsilon}, \tag{12}$$

and compute the pairwise cosine similarity of normalised *attention-pattern rows*:

$$\cos_{\text{attn}}(i,j) \;=\; \langle \tilde{A}_{i,\cdot},\, \tilde{A}_{j,\cdot} \rangle, \qquad i \neq j, \quad i,j \in \text{same chunk}. \tag{13}$$

For each token i we retain the top-$k_{\text{intra}} = 8$ neighbours by similarity, and we add the corresponding edge (i, j, w_{ij}) with $w_{ij} = \cos_{\text{attn}}(i,j)$ only if $w_{ij} > \tau_{\text{intra}} = 0.3$. The intuition is that two tokens with similar attention patterns are functionally related: they participate in the same query-driven content. This is distinct from cosine similarity of raw key vectors; we use the attention-pattern formulation because it captures *role* similarity (which queries call on this token) rather than mere representational similarity.

Cross-chunk edges from raw attention. For each chunk after the first, we read the per-query attention from queries in the current chunk to keys in all earlier chunks:

$$\bar{A}^{\text{cross}}_{j,i} \;=\; \frac{1}{H}\sum_{h=1}^{H} A^{(1)}_{h,j,i}, \qquad j \in \text{current chunk},\; i \in \text{earlier chunk}. \tag{14}$$

For each query j we retain the top-$k_{\text{cross}} = 4$ source positions i by $\bar{A}^{\text{cross}}_{j,i}$, and we add the edge (i, j, w_{ij}) with $w_{ij} = \bar{A}^{\text{cross}}_{j,i}$ only if $w_{ij} > \tau_{\text{cross}} = 0.02$. The lower threshold reflects the lower numerical scale of softmax attention compared to row-normalised cosine; tuning is conservative.

Edge set.

$$E \;=\; E_{\text{intra}} \cup E_{\text{cross}}, \qquad |E| \;=\; O(n), \tag{15}$$

since each token contributes at most $k_{\text{intra}} + k_{\text{cross}}$ edges. The cost of *producing* E is dominated by materialising the cross-chunk attention block before the top-k_{cross} selection: this is $O(n^2)$ in the present CPU-NumPy implementation and is the engineering bottleneck quantified in Section 7.

First layer only. We extract attention from the first layer alone. Empirically the first layer's attention pattern already discriminates content tokens from filler with high contrast, and using only the first layer avoids the $O(L)$ cost of cross-layer aggregation and avoids the question of how to merge attention from layers with differing semantics. We treat "first layer" as a deliberate algorithmic choice; ablating it would require replacing it with another single layer or a learned aggregation, which we do not do here.

3.3 Stage S2: Trunk Construction

A *trunk* is a contiguous group of tokens treated as a single eviction unit. Trunks are constructed in three steps.

Step 1: sentence-boundary segmentation. We split the token sequence at the tokenizer ids corresponding to ".", "**!**", "?", and "\n". A sentence-ending token is included as the final token of the sentence it terminates. This yields an initial sequence of segments $s_1, s_2, \ldots, s_M$ at $O(n)$ cost.

Step 2: cumulative co-attention-driven merge (single left-to-right pass). Sentence boundaries are too fine for many documents. We grow trunks left-to-right by checking whether each successive sentence merges with the trunk currently being built. For the running trunk g_{cur} and the next sentence s_i, define the interface window

$$P \;=\; \text{last5}\big(g_{\text{cur}}\big) \;\times\; \text{first5}(s_i) \tag{16}$$

and the subset of edges in E whose endpoints fall in P:

$$E_P \;=\; \big\{(a,b,w_{ab}) \in E : (a,b) \in P\big\}. \tag{17}$$

The cross-interface co-attention strength is the average weight over *recorded* edges only:

$$\text{CAS}\big(g_{\text{cur}}, s_i\big) \;=\; \begin{cases} \dfrac{1}{|E_P|}\displaystyle\sum_{(a,b,w)\in E_P} w & \text{if } |E_P| > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

Sentence s_i is merged into the running trunk iff

$$\mathrm{CAS}\big(g_{\mathrm{cur}}, s_i\big) > \tau_{\mathrm{merge}} \quad \text{and} \quad |g_{\mathrm{cur}}| + |s_i| \le T_{\max}, \tag{19}$$

with defaults $\tau_{\mathrm{merge}} = 0.3$ and $T_{\max} = 32$. If the merge condition holds we set $g_{\mathrm{cur}} \leftarrow g_{\mathrm{cur}} \cup s_i$; otherwise we commit g_{cur} as a finalised trunk and start a new running trunk from s_i. The pass is single and left-to-right; we do not iterate to convergence.

Averaging over recorded edges only (rather than over all 25 candidate pairs in P) is important: most interface pairs carry no edge, so dividing by 25 would systematically dilute genuine strong links and bias the algorithm against merging.

Step 3: maximum-size split. After Step 2 a trunk may still exceed $T_{\max}$ (a long unbroken code block, an enumeration with no sentence punctuation). For any such trunk we split uniformly into $\lceil |g|/T_{\max} \rceil$ pieces of approximately equal size, guaranteeing a hard upper bound on per-trunk size and bounding the trunk graph's size at $O(n/T_{\max})$.

Comparison. ChunkKV's fixed 10-token chunks have purely positional boundaries unrelated to linguistic structure. EpiCache's K-Means over sentence embeddings requires an external embedding model and is structurally tied to multi-turn conversational data. Our merge step is local, uses only the model's own first-layer attention through E, and is conditioned on the prefill—so the same input text under two different prompts can produce two different trunkings.

3.4 Stage S3: Per-Token Encoding Impact M_i

The token-level encoding impact M_i captures the intrinsic salience of token i at the moment it enters the cache. It is computed once at the end of prefill (after S1 has produced S_i and before S2 builds trunks) and is not updated during decode.

Inputs. S1 has provided S_i from Equation (11), clipped into $[M_{\min}, M_{\max}]$. From the input ids alone we also compute, in $O(n)$ time, the per-token frequency

$$c_i \;=\; \big|\big\{j \in \{0, \dots, n-1\} : \mathrm{id}(j) = \mathrm{id}(i)\big\}\big|, \tag{20}$$

the number of times the surface token id at position i appears in the current context.

Von Restorff rarity. We instantiate the Von Restorff distinctiveness effect [12] as

$$U_i \;=\; \frac{1}{1 + \log(1 + c_i)} \;\in\; (0, 1]. \tag{21}$$

Concrete values: $U_i(1) \approx 0.591$, $U_i(10) \approx 0.295$, $U_i(100) \approx 0.178$. The $1/\log$ form has three properties we sought: bounded in $(0, 1]$ without a clip, slow decay so that the contrast between "appears once" and "appears five times" is large but the contrast between "appears 100 times" and "appears 500 times" is small, and strictly attention-independent.

Final M_i. S_i and U_i live on incommensurable scales (S_i is unbounded above $M_{\max}$ before clipping; $U_i \in (0, 1]$). We normalise S_i to $[0, 1]$, mix equally with U_i, and rescale back to the salience scale:

$$\widetilde{S}_i \;=\; \frac{S_i}{M_{\max}}, \qquad M_i \;=\; \mathrm{clip}\Big(M_{\max} \cdot \big(\tfrac{1}{2}\widetilde{S}_i + \tfrac{1}{2}U_i\big),\; M_{\min},\; M_{\max}\Big). \tag{22}$$

The equal-weight prior is deliberately uninformative; the ablation in Section 6 also evaluates the configurations $M_i = \widetilde{S}_i \cdot M_{\max}$ (no Von Restorff) and $M_i = 1$ (uniform), both of which underperform the equal-weight default. The Von Restorff term is the single most impactful component, with removal causing a -0.383 absolute drop on Llama-3.1-8B Needle at $\beta = 0.5$.

Trunk-level summary. At trunk-creation time (S2, immediately after Step 3), each trunk g is assigned a fixed per-trunk encoding impact equal to the mean of the top-3 member token M_i values:

$$\overline{M_i}(g) \;=\; \frac{1}{\min(3, |g|)} \sum_{i \,\in\, \mathrm{Top3}_{M_i}(g)} M_i. \tag{23}$$

$\overline{M_i}(g)$ is stored as a per-trunk constant from this point on; the score composition in Section 3.6 reads it directly without recomputation. Averaging the top-3 rather than all member tokens prevents a few high-impact tokens from being diluted by surrounding stop words.

3.5 Stage S4: Per-Trunk Crystallization Score D

The trunk-level score $D(g)$ is the second, structural dimension of the eviction score. It corresponds to "structural extent" in the Crystallization Memory Framework [11]: a trunk is structurally central if it is densely connected to many other trunks via co-attention.

Trunk-level edge weight (with prune threshold). For a pair of distinct trunks (g_a, g_b), collect the cross-trunk edges as a multiset of weights:

$$W_{ab} \;=\; \big\{w_{ij} : (i,j,w_{ij}) \in E,\; i \in g_a,\; j \in g_b\big\} \;\cup\; \big\{w_{ji} : (j,i,w_{ji}) \in E,\; i \in g_a,\; j \in g_b\big\}. \tag{24}$$

The trunk-level edge weight is the mean weight scaled by the square root of the empirical edge density:

$$W(g_a, g_b) \;=\; \begin{cases} \overline{w}_{ab} \cdot \sqrt{\dfrac{|W_{ab}|}{|g_a| \cdot |g_b|}} & \text{if it exceeds } \tau_{\text{edge}} = 0.05, \\ 0 & \text{otherwise (edge pruned),} \end{cases} \tag{25}$$

where $\overline{w}_{ab}$ is the arithmetic mean of W_{ab}. The mean-weight factor prevents a pair of trunks linked by many low-weight edges from outscoring a pair linked by fewer high-weight edges; the square-root density factor prevents the weight from collapsing toward zero when one trunk is large, while still penalising sparse connectivity. The prune threshold τ_{edge} keeps the trunk graph sparse for downstream maintenance during the decode-time breath cycle (Section 3.8); at prefill time it has only a small effect on $\deg(g)$.

Weighted degree. The weighted degree of trunk g is the sum of its surviving trunk-level edge weights:

$$\deg(g) \;=\; \sum_{g' \neq g} W(g, g'). \tag{26}$$

Sigmoid mapping to $[0,1]$. $\deg(g)$ is heavy-tailed and scale-dependent. We standardise it across the trunks of the current sequence and pass through a logistic:

$$\mu \;=\; \frac{1}{m}\sum_{g} \deg(g), \qquad \sigma \;=\; \operatorname{std}\big(\{\deg(g)\}\big), \qquad D(g) \;=\; \sigma_{\text{logistic}}\!\left(s \cdot \frac{\deg(g) - \mu}{\max(\sigma, \varepsilon)}\right), \tag{27}$$

where $\sigma_{\text{logistic}}(x) = (1 + e^{-x})^{-1}$ and $s = 5.0$ is a steepness parameter (the $\max(\sigma, \varepsilon)$ guards against degenerate $\sigma \approx 0$ on near-uniform inputs; we set $\varepsilon = 10^{-8}$ and fall back to $\sigma = 1$ when $\sigma < \varepsilon$). The sigmoid offers high resolution near the median (where most trunks live) and saturates in the extremes, preventing a single ultra-central trunk from dominating the score scale. The choice $s = 5.0$ places the bulk of the trunk distribution in the resolved interior of the sigmoid; the dependence on s is mild over $s \in [2, 10]$.

3.6 Stage S5: Two-Path Score Composition

Given the per-trunk $D(g)$ from Equation (27) and the per-trunk $\overline{M_i}(g)$ from Equation (23), we compose the score that drives eviction.

Composition by maximum. Inside the eviction routine, the $\overline{M_i}$ side is log-compressed and min–max-normalised over the unprotected trunks (the set $\mathcal{A}$ defined in Section 3.7). Letting $\ell(g) = \log\big(1 + \overline{M_i}(g)\big)$,

$$\widetilde{M_i}(g) \;=\; \frac{\ell(g) - \min_{g' \in \mathcal{A}} \ell(g')}{\max_{g' \in \mathcal{A}} \ell(g') - \min_{g' \in \mathcal{A}} \ell(g') + \varepsilon} \;\in\; [0,1], \tag{28}$$

and the composite per-trunk score is

$$\text{score}(g) \;=\; \max\Big(D(g),\;\; \alpha \cdot \widetilde{M_i}(g) \Big), \tag{29}$$

with $\alpha = 1.0$ by default. The logarithm compresses the dynamic range of $\overline{M_i}$ before normalisation, preventing a single extreme outlier trunk from squashing the rest of the distribution toward zero.

Why $\max$, not weighted average? A weighted average $\mathrm{score}(g) = \mu\, D(g) + (1-\mu)\, \widetilde{M}_i(g)$ penalises a trunk that scores high along one dimension and low along the other. But the framework predicts exactly two such trunk classes should be retained, for two different reasons:

- *High D, low M_i:* a structurally central section header or topic anchor that is itself unremarkable token-by-token.
- *Low D, high M_i:* an isolated but uniquely encoded fact (a single Needle, a one-time proper noun) that is not connected to the rest of the document.

Weighted averaging actively suppresses both classes by their weak side. With $\max$, each dimension acts as an independent survival path: a trunk excelling along D survives whether or not it also excels along M_i, and vice versa. This is structurally distinct from DefensiveKV's $\max$ over multiple noisy observations of the *same* attention signal (a within-dimension robustifier) and from CAOTE's closed-form fusion of attention with value into a *single* scalar (a within-token signal fusion).

3.7 Stage S6: Branch Dissolution Eviction

Given the per-trunk scores from Equation (29) and a budget, we apply a two-stage procedure: proportional retention across trunks, then M_i-ranked retention within each partially retained trunk.

Budget and protected set. We always retain (i) the first $N_{\text{sink}} = 4$ tokens (attention sinks; cf. [3]) and (ii) the last $W_{\text{recent}} = 128$ tokens (recent window). A trunk is marked protected if any of its member positions lies in $[0, N_{\text{sink}})$ or $[n - W_{\text{recent}}, n)$; protected trunks are removed from the eviction pool in full. To guarantee the protected set fits in the cache, the effective budget is floored:

$$B \;=\; \max\big(N_{\text{sink}} + W_{\text{recent}},\; \lceil \beta n \rceil\big). \tag{30}$$

Let $\mathrm{Protected}$ denote the set of protected trunks and $\mathcal{A}$ the set of unprotected (alive) trunks. Define

$$B_{\text{prot}} \;=\; \sum_{g \,\in\, \mathrm{Protected}} |g|, \qquad B_{\text{avail}} \;=\; B - B_{\text{prot}}, \qquad T_{\text{alive}} \;=\; \sum_{g \,\in\, \mathcal{A}} |g|, \tag{31}$$

and $c_0 = \max(0, T_{\text{alive}} - B_{\text{avail}})$, the number of tokens that must be removed from the unprotected pool. If $c_0 = 0$, no eviction is performed.

Stage 1: bottom-up proportional allocation with overshoot-on-degenerate-partial. Sort the unprotected trunks by $\mathrm{score}(g)$ in ascending order. Initialise $r_g \leftarrow 1$ for all $g \in \mathcal{A}$ and $c \leftarrow c_0$. Iterate over the sorted trunks from lowest score upward and update r_g as follows:

$$\big(r_g,\, c\big) \;\leftarrow\; \begin{cases} (1,\, c) & \text{if } c \le 0, \text{ break out of loop,} \\ (0,\, c - |g|) & \text{if } |g| \le c, \text{ (fully evict),} \\ \big(\frac{|g| - c}{|g|},\, 0\big) & \text{if } |g| > c \text{ and } |g| - c \ge N_{\min}, \text{ (partial retention),} \\ (0,\, c - |g|) & \text{if } |g| > c \text{ and } |g| - c < N_{\min}, \text{ (degenerate partial: overshoot).} \end{cases} \tag{32}$$

The fourth branch (*degenerate partial*) is the minimum-survival rule: rather than retain fewer than $N_{\min} = 3$ tokens of a trunk—which would not constitute a coherent fragment—we drop the entire trunk and overshoot the cut by $|g| - c < N_{\min}$ tokens. The overshoot is bounded in absolute value by $N_{\min} - 1 = 2$ tokens and the loop terminates at the next iteration since c is now negative.

Stage 2: intra-trunk M_i-ranked selection. For each trunk g with $0 < r_g < 1$, the number of tokens to keep is

$$k_g \;=\; \min\Big(|g|,\; \max\big(N_{\min},\; \mathrm{round}(r_g \cdot |g|)\big)\Big), \tag{33}$$

where round denotes banker's rounding. We sort the tokens of g by per-token M_i in descending order and retain the top k_g. This is the only stage that breaks contiguity within a trunk; protected trunks and trunks with $r_g = 1$ remain contiguous.

Algorithm 1 Branch dissolution eviction (matches `TrunkTable.branch_dissolve` in `crystalcache/core/trunk.py`).

Require: trunks $\{g_1, \dots, g_m\}$ with $D(g_k)$ and $\overline{M_i}(g_k)$; per-token M_i array; budget B
Mark trunks containing a token in $[0, N_{\text{sink}})$ or $[n - W_{\text{recent}}, n)$ as protected
Compute $B_{\text{prot}}, B_{\text{avail}}, T_{\text{alive}}$ via Eq. (31); $c \leftarrow \max(0, T_{\text{alive}} - B_{\text{avail}})$
if $c = 0$ **then**
    return no-op (no eviction needed)
end if
Compute $\widetilde{M}_i(g)$ via Eq. (28) over unprotected trunks $\mathcal{A}$
Compute $\text{score}(g) = \max\big(D(g), \alpha\, \widetilde{M}_i(g)\big)$ for each $g \in \mathcal{A}$
Sort $\mathcal{A}$ ascending by score; initialise $r_g \leftarrow 1$ for all $g \in \mathcal{A}$
for each $g \in \mathcal{A}$ in sorted order **do**
    update (r_g, c) via Eq. (32)
    if $c \leq 0$ after update **then**
        break
    end if
end for
$\mathcal{R} \leftarrow$ indices of all tokens in protected trunks
for each $g \in \mathcal{A}$ **do**
    if $r_g \geq 1$ **then**
        $\mathcal{R} \leftarrow \mathcal{R} \cup g$
    else if $r_g \leq 0$ **then**
        $D(g) \leftarrow 0$ ▷ Eq. (34)
    else
        compute k_g via Eq. (33)
        sort tokens of g by per-token M_i descending; let T_g be the top k_g
        $\mathcal{R} \leftarrow \mathcal{R} \cup T_g$; $D(g) \leftarrow r_g \cdot D(g)$ ▷ Eq. (34)
    end if
end for
compress $K^{(\ell)}, V^{(\ell)}$ via Eq. (35) for each layer ℓ
return $\mathcal{R}$

Crystallization side-effects. Eviction also updates the trunk's D score in place, modelling the structural cost of branch loss:

$$D(g) \;\leftarrow\; \begin{cases} 0 & \text{if } r_g = 0 \text{ (fully evicted)}, \\ r_g \cdot D(g) & \text{if } 0 < r_g < 1 \text{ (partial)}, \\ D(g) & \text{if } r_g = 1 \text{ (unchanged)}. \end{cases} \tag{34}$$

This is consequential during the decode-time breath cycle (Section 3.8): a partially dissolved trunk enters the next breath cycle with a proportionally weakened D and is therefore more vulnerable to further eviction unless its remaining tokens are reactivated.

Cache compression. Let $\mathcal{R} \subseteq \{0, 1, \dots, n-1\}$ be the union of all retained token indices and $\mathcal{D} = \{0, \dots, n-1\} \setminus \mathcal{R}$ the evicted indices. For each layer ℓ and each KV head we apply

$$K^{(\ell)} \;\leftarrow\; K^{(\ell)}[\mathcal{R}], \qquad V^{(\ell)} \;\leftarrow\; V^{(\ell)}[\mathcal{R}], \tag{35}$$

implemented as a single `torch.index_select` per tensor. The position-id tensor used by RoPE-based attention during decode is rebuilt from $\mathcal{R}$ so that absolute positions of retained tokens are preserved (we do not renumber). Algorithm 1 summarises the full procedure as it is realised in `TrunkTable.branch_dissolve`.

3.8 Decode-Time Mechanism (Reserved for Long Generation)

We now describe the decode-time portion of CrystalCache. *This mechanism is implemented but is not exercised in the experiments of Section 5*, since our maximum generation length of 50 tokens is shorter than the breath interval $T_{\text{breath}} = 64$. We report it in full for completeness and because future work explicitly targets long-generation evaluation.

Per-step M_i-modulated D decay. After each decode step's forward pass, every trunk's D score is decayed at a rate modulated by the trunk's own D and $\overline{M_i}$:

$$\rho(g) \;=\; \beta_0 \cdot \exp\!\left(-\frac{D(g)\cdot \overline{M_i}(g)}{D_c}\right), \qquad D(g) \;\leftarrow\; \max\!\big(0,\; D(g) - \rho(g)\cdot D(g)\cdot \Delta t\big), \tag{36}$$

with $\beta_0 = 0.05$, $D_c = 2.0$, and $\Delta t = 1$. The $\exp(-D \cdot \overline{M_i}/D_c)$ factor is self-stabilising: trunks that already have high D or high $\overline{M_i}$ decay slowly, while trunks with low D and low $\overline{M_i}$ decay rapidly. This is the dissolution side of the crystallization analogy.

Breath cycle (every $T_{\text{breath}} = 64$ steps). Every 64 decode steps, four substeps run in sequence.

(a) Activation detection. For each trunk g, compute the mean inverse L_2 norm of the keys of its tokens (using only the first layer); normalise by the per-sequence maximum to obtain $\eta \in [0,1]^n$. A trunk is flagged *active* if more than half of its member tokens have $\eta > \theta_{\text{act}}$, with $\theta_{\text{act}} = 0.3$.

(b) Resonance broadcast. A multi-hop BFS originates at each active trunk and propagates a pulse of strength 1 along trunk-graph edges. After h hops the pulse is multiplicatively attenuated by the product of edge weights and a per-hop decay factor:

$$\text{pulse}_h(g') \;=\; \text{pulse}_{h-1}(g) \cdot W(g,g') \cdot \gamma_{\text{hop}}, \tag{37}$$

where $\gamma_{\text{hop}} = 0.5$. The BFS terminates at $h = 2$ (the default $\max$-hops) or when the pulse falls below $\tau_{\text{pulse}} = 0.01$. A trunk that receives pulse p has its D raised by

$$D(g') \;\leftarrow\; \min\!\big(1,\; D(g') + \alpha_{\text{res}} \cdot p \cdot (1 - D(g'))\big), \tag{38}$$

with $\alpha_{\text{res}} = 0.15$. The factor $(1-D)$ produces diminishing returns near saturation. This implements semantic spreading activation: a re-activated trunk pulls structurally related trunks back from dissolution.

(c) Hebbian edge update (additive, capped). For pairs of trunks (g, g') both flagged active in the same breath cycle, the trunk-graph edge weight is reinforced additively:

$$W(g,g') \;\leftarrow\; \min\!\big(1,\; W(g,g') + \eta_{\text{Hebb}}\big), \tag{39}$$

with $\eta_{\text{Hebb}} = 0.05$. For all other edges, multiplicative decay applies:

$$W(g,g') \;\leftarrow\; \gamma_{\text{decay}} \cdot W(g,g'), \tag{40}$$

with $\gamma_{\text{decay}} = 0.99$. Edges falling below $\tau_{\text{edge}} = 0.05$ (Eq. (25)) are pruned. This implements "trunks frequently co-active during decoding become more structurally bound, while inactive bonds fade."

(d) Re-eviction. The budget B is re-evaluated against the updated cache (which may have grown by up to T_{breath} new decode tokens), Algorithm 1 is re-run on the updated $\text{score}(g)$ values with $\text{Protected} = \emptyset$, and the cache is recompressed via Equation (35). Note that during decode the protected set is empty; the sink and recent-window tokens are not re-protected because the algorithm is meant to allow them to age naturally if not re-activated.

Status in this work. None of (a)–(d) fires in the experiments of Section 5. The mechanism is best understood as a design extension; we return to its empirical validation as the principal item of future work in Section 8.

3.9 Computational Complexity

Let n be the prefill sequence length, L the number of layers, H the number of attention heads, $C = 1024$ the chunk size, and $m \approx n/T_{\max}$ the number of trunks. Table 3 summarises the per-stage cost; we then comment on the bottleneck.

The bottleneck is engineering, not algorithmic. At 32K tokens the cross-chunk top-k stage of S1 takes approximately 20 s on an H100 NVL (Section 7), exceeding the model's own forward pass at the same length (14.7 s). It must dominate at large n relative to anything sub-quadratic, since it is $O(n^2)$, but the absolute constant is much higher than necessary because (i) the implementation is CPU NumPy rather than GPU CUDA, and (ii) we currently materialise the full cross-chunk attention block before selecting the top-k, whereas approximate top-k via locality-sensitive hashing or a sparse attention sketch would avoid materialising the dense block at all. We treat the GPU port and LSH approximation as engineering work that does not affect the algorithm's correctness, only its wall-clock cost. Section 7 quantifies the overhead at 4K, 8K, 16K, and 32K.

Table 3: Per-stage computational cost of CrystalCache prefill, expressed in big-O. The standard prefill row is the cost of the underlying model and is unchanged by CrystalCache.

Stage	Cost	Comment
Standard prefill forward	$O(n^2LHd)$	unchanged
S1 (S_i aggregation)	$O(nH)$	one pass over first-layer attention
S1 (intra-chunk row cosine)	$O(nC)$	cosine within $\lceil n/C \rceil$ chunks
S1 (cross-chunk top-k)	$O(n^2)$	current bottleneck (CPU NumPy); see below
S2 (sentence split + merge)	$O(n + m\,k_{\text{intra}})$	local merging via interface windows
S3 (U_i, mixture, trunk $\overline{M_i}$)	$O(n)$	token-frequency counter + arithmetic
S4 (trunk graph + D)	$O(m^2)$	dense weighted degree; $m \ll n$
S5 (score composition)	$O(m)$	per-trunk arithmetic
S6 (branch dissolution)	$O(m \log m + n)$	sort + index gather

Decode-time cost. After eviction the cache has size $B \approx \lceil \beta n \rceil$, so each decode step costs $O(BLHd)$ rather than $O(nLHd)$. At $\beta = 0.5$ each step is $2\times$ cheaper; at $\beta = 0.3$, $\sim 3.3\times$ cheaper. Steady-state KV memory is reduced by exactly the same factor; this is the user-facing benefit of the method (Section 7). The decode-time breath cycle adds an $O(m)$ activation pass plus a bounded BFS of cost $O(\bar{d}^2)$ where $\bar{d}$ is the mean trunk-graph degree, and is amortised over $T_{\text{breath}} = 64$ steps; at typical m this overhead is sub-millisecond.

4 Experimental Setup

This section describes the models, tasks, baselines, and protocol used for all experiments in Section 5, the ablations in Section 6, and the cost measurements in Section 7. Hyperparameter defaults for CrystalCache are those of Table 2 (corresponding to `configs/default.yaml`); deviations are noted explicitly.

4.1 Models

We evaluate on three open-weight instruction-tuned models spanning three different training pipelines, attention architectures, and positional encodings. The diversity is deliberate: a method that wins only on Llama is hard to attribute to its design rather than to favourable interaction with one model's idiosyncrasies.

Table 4: Models used in this paper. All weights are loaded in bf16. "Attn" denotes the attention pattern; "RoPE base" is the rotary positional encoding base.

Model	L	H	H_{kv}	d	Attn	RoPE base
Llama-3.1-8B-Instruct [16]	32	32	8	128	GQA	5×10^5
Mistral-7B-Instruct-v0.3 [17]	32	32	8	128	GQA	1×10^6
Qwen3-8B [18]	36	32	8	128	GQA	1×10^6

All three models are loaded with HuggingFace `transformers` (≥ 4.45) at default precision (bf16). The base attention implementation during prefill is SDPA; CrystalCache temporarily switches the first-layer attention to eager during prefill in order to extract the attention tensor (Section 3.2), then reverts to SDPA. Baselines do not need this switch and run entirely under SDPA.

4.2 Tasks

We evaluate on two long-distance retrieval tasks for the headline result, and on the LongBench v1 suite for a broader-coverage check.

Needle-in-a-Haystack (NIAH) [5]. A single critical fact (the "needle") is inserted at a controlled depth into a long filler passage. The model is then asked a question whose answer is the needle. Five needle templates are sampled uniformly per construction, each instantiated with a random four-digit value ("the special magic number is 7492," "the secret code for Project Aurora is …", etc.) so that no test answer can be recovered from the model's pretraining priors.

Configuration. Context lengths $\{4096, 8192, 16384, 32768\}$ tokens, needle depths $\{0.0, 0.25, 0.5, 0.75, 1.0\}$ as fractions of context, 3 repetitions per (length, depth) cell, for a total of $4 \times 5 \times 3 = 60$ samples. The metric is exact-match accuracy of the four-digit value in the generated output.

Delayed Association (DA). A diagnostic benchmark we constructed specifically to probe the contribution of D (associative crystallization). The structure is: a critical fact about a topic X is placed early in the context ("the secret code for Project Aurora is 7492"), followed by thousands of filler tokens interspersed with related-but-non-redundant mentions of X ("progress on Aurora was reviewed," "the Aurora team adjusted resource allocation," etc.). The mentions do *not* repeat the fact—they only sustain semantic association with the topic—and the model is then asked the original question.

Configuration. Three topic templates (Project Aurora, Agent Nightingale, Formula X), distances of $\{4096, 8192, 16384\}$ tokens between the fact and the question, two density regimes (high: four related mentions per topic; low: two), and 10 samples per (distance, density) cell, for a total of $3 \times 2 \times 10 = 60$ samples. Each sample uses one of the three topic templates randomly. The metric is exact-match of the value.

Why this benchmark. On NIAH the relevant fact is mentioned exactly once, so the score that distinguishes a Needle's surrounding trunk from a filler trunk is dominated by M_i (specifically by the rarity term U_i). On DA the fact is also mentioned exactly once, but the surrounding context contains *related* mentions that establish co-attention edges to the fact's trunk. A method that retains the fact through these structural connections (high D even though the fact's M_i is moderate) should outperform a method that retains tokens only by direct attention statistics. Section 6 confirms this: the D*-only* ablation underperforms the full method on DA by a smaller margin than on NIAH, and the M_i*-only* ablation does the opposite.

LongBench v1 [13]. A standard real-world long-context benchmark. We use the English subsets only:

> `narrativeqa`, `qasper`, `multifieldqa_en`, `hotpotqa`, `2wikimqa`, `musique`, `trec`, `triviaqa`, `samsum`, `passage_count`, `passage_retrieval_en`.

11 subsets total, 20 samples per subset (capped from each subset's full size), for 220 samples per (model, method, β) cell. Each subset has its own scoring metric (F1, ROUGE-L, accuracy, edit distance), as specified by the original LongBench evaluation protocol; we report the official per-subset score and the mean over subsets.

To keep prefill time bounded we cap input context at 20,000 words per sample (truncating from the middle, preserving the prompt's prefix and suffix). The same cap is applied uniformly across all methods including FullCache, so the comparison remains fair.

4.3 Baselines

We compare CrystalCache against a representative set of first- and second-generation eviction methods, plus the no-eviction upper bound. All baselines are reimplemented in our harness from their published descriptions and run under identical model/tokenizer/decoding settings; only the cache-management policy differs.

Llama-3.1-8B (full baseline suite).

- **FullCache.** No eviction; serves as the upper bound on accuracy and the reference for memory savings.
- **StreamingLLM [3].** First $N_{\text{sink}} = 4$ tokens plus the most recent W_{recent} tokens; everything in between is discarded. Window size is set so that total retained tokens equal the budget.
- **H2O [1].** Cumulative-attention heavy hitters plus recent window.
- **SnapKV [2].** Per-head observation-window selection at the end of prefill.
- **PyramidKV [15].** Layer-wise budget allocation (more KV in early layers, less in late layers) on top of cumulative-attention scoring.
- **ChunkKV [6].** Fixed 10-token chunk grouping with intra-chunk attention sum scoring.

Mistral-7B and Qwen3-8B (compute-constrained subset). H2O, SnapKV, and ChunkKV only. We omit StreamingLLM and PyramidKV for these models because the Llama results already establish that StreamingLLM is far below the contemporary baselines and that PyramidKV's layer-wise budget is orthogonal to the score, so neither would change our central comparison; eliminating them halved the experiment compute. FullCache for Mistral and Qwen3 was not run for the headline tables (each FullCache run on Needle 32K alone takes ~ 1.5 hours of H100 time) but the achievable upper bound is structurally bounded by what the underlying model can solve at zero eviction.

Other recent works. We do not include CAOTE [8], EpiCache [7], or DefensiveKV [9] in the headline tables. The reasons are pragmatic rather than evaluative: at the time of writing, CAOTE's reference implementation requires

modifications to per-layer attention modules that are not ABI-compatible with the kv-press style harness we use for the others; EpiCache's design is structurally tied to multi-turn conversational data with explicit utterance boundaries and would require heuristic adaptation to general documents; DefensiveKV's reference release post-dates our experiment freeze. We discuss the conceptual relationship in Section 2.5 and treat their head-to-head empirical comparison as future work in a unified harness.

4.4 Budgets

All non-FullCache methods are evaluated at two retention budgets:

$$\beta \in \{0.3, 0.5\}. \tag{41}$$

At $\beta = 0.5$ the cache is halved (yielding $\sim 50\%$ steady-state KV memory savings); at $\beta = 0.3$ it is reduced by $\sim 70\%$. Both methods use the same effective B as defined by Equation (30) (i.e. floor at $N_{\text{sink}} + W_{\text{recent}}$); for all our experiments the floor is far below βn at the context lengths used, so the floor is inactive.

We deliberately do not evaluate at extreme low budgets ($\beta \leq 0.2$). At those budgets the residual cache is small enough that no method, including FullCache-with-truncation as a sanity check, recovers the answer reliably on most NIAH and DA samples; the comparison becomes dominated by the budget floor and the protected set rather than by any scoring strategy. Robustness at extreme low budgets is listed as future work in Section 8.

4.5 Decoding

All methods use *greedy* decoding (argmax of the next-token logits). Sampling is disabled to remove a source of cross-method variance. Maximum generation length is 50 tokens for NIAH and DA; LongBench uses each subset's official `max_gen` as published in [13].

We chose greedy over sampling for two reasons. First, the metric on NIAH and DA is exact-match of a four-digit value, so any non-determinism would directly inflate variance. Second, sampling is orthogonal to the cache-management policy under test; we want the comparison to reflect only the eviction algorithm, not interactions between sampling temperature and the residual cache.

Note on the dormant decode-time mechanism. The maximum generation length of 50 tokens is shorter than the breath interval $T_{\text{breath}} = 64$ (Section 3.8), so the breath cycle does not fire in any reported experiment. CrystalCache reduces, in this regime, to the prefill pipeline (S1–S6) followed by static decoding on the compressed cache. The empirical validation of the breath cycle, M_i-modulated D decay, multi-hop resonance, and Hebbian edge updates is deferred to future work that targets long-generation evaluation (Section 8).

4.6 Hardware and Implementation

Hardware. All experiments run on a single NVIDIA H100 NVL (94 GB HBM3) in single-GPU configuration. CPU is an Intel Xeon Platinum (32 cores). Disk I/O does not bottleneck any experiment; all model weights fit in HBM at bf16, and the LongBench subsets fit in CPU RAM.

Software. PyTorch $\geq 2.4.0$, `transformers` $\geq 4.45.0$, `accelerate` $\geq 0.34.0$, NumPy ≥ 1.26, NetworkX ≥ 3.2 (for the trunk graph). FlashAttention is available but not used in the eager-mode portion of CrystalCache prefill (eager attention is required to materialise the first-layer attention tensor). Baselines run under SDPA throughout.

CrystalCache implementation status. The current implementation of the co-attention edge extraction (Section 3.2) is CPU NumPy, not GPU CUDA. This is the dominant cost at 32K context (Section 7). A GPU port and an LSH-based approximate top-k are listed as future engineering work that does not affect the algorithm's correctness, only its wall-clock time.

4.7 Experimental Protocol and Reproducibility

Per-experiment isolation. Each (model, method, benchmark, budget) cell is run in its own Python subprocess. This is necessary because some baselines retain CUDA state between runs in ways that interact unpredictably with the kv-press-style harness; running each cell in a fresh process eliminates this confound at the cost of repeated model loading. All cells use the same loaded model (cached on disk) and the same tokenizer.

Table 5: Needle-in-a-Haystack: exact-match accuracy at retention budgets $\beta = 0.3/0.5$. The Llama row uses the full baseline suite; Mistral and Qwen3 use the compute-constrained subset (Section 4.3). FullCache is the no-eviction upper bound for Llama; for Mistral and Qwen3 the upper bound was not measured for headline reporting reasons. Bold marks the best non-FullCache value per column.

Method	Llama-3.1-8B	Mistral-7B	Qwen3-8B
FullCache (upper bound)	1.000 / 1.000	—	—
StreamingLLM [3]	0.267 / 0.233	—	—
H2O [1]	0.633 / 0.467	0.150 / 0.267	0.117 / 0.100
SnapKV [2]	0.633 / 0.500	0.300 / 0.233	0.133 / 0.167
PyramidKV [15]	0.250 / 0.417	—	—
ChunkKV [6]	0.683 / 0.583	0.367 / 0.350	0.133 / 0.083
CrystalCache	**0.767 / 0.767**	**0.433 / 0.483**	**0.350 / 0.333**

Random seeds. Every benchmark uses a fixed seed (`42`) for sample construction so that the same NIAH or DA samples are presented to every method on every run. Greedy decoding eliminates within-method stochasticity. Reported metrics are therefore deterministic given the model, method, and budget; we do not report error bars across seeds, as no within-cell variance exists to bar.

Compute budget. The full $3 \times$ (method count) $\times\ 2 \times 3 = 90$+ cell sweep across NIAH, DA, and LongBench at $\beta \in \{0.3, 0.5\}$ totalled approximately 80 H100-hours, dominated by FullCache and CrystalCache runs at 32K NIAH (where prefill alone is ~ 90 seconds per sample).

Fairness controls. For every (model, benchmark, budget) cell we use identical inputs across methods, identical tokenization, identical greedy decoding, identical `max_new_tokens`, identical input truncation policy (when applicable for LongBench), and identical SDPA-vs-eager attention conventions on the non-CrystalCache forward passes. The only thing that varies is the cache-management policy.

Code and artefacts. All code, configurations, and raw per-sample outputs (the JSON files behind every metric reported in Section 5) are released alongside this paper. The exact CrystalCache configuration is `configs/default.yaml`; baseline configurations are documented in

5 Main Results

This section reports CrystalCache's performance against the baselines of Section 4.3 on the three benchmarks of Section 4.2. Section 5.1 reports Needle-in-a-Haystack across all three models. Section 5.2 reports Delayed Association. Section 5.3 reports LongBench v1 honestly, including the regime where CrystalCache is not the strongest method. Section 5.4 consolidates the cross-model consistency claim from Section 2.5, and Section 5.5 examines per-context-length and per-depth structure of the Needle result.

All numbers in this section are extracted directly from the raw `metrics.json` artefacts produced by the experimental harness. Per-sample outputs are released alongside the paper.

5.1 Needle-in-a-Haystack

Table 5 reports exact-match accuracy on the 60-sample NIAH evaluation across all three models and both budgets. CrystalCache wins all six (model, budget) cells.

Three observations.

Absolute gains are large. On Llama at $\beta = 0.5$, CrystalCache reaches 0.767 versus the strongest baseline ChunkKV at 0.583, a $+0.184$ absolute improvement (equivalently, $+31.6\%$ relative). On Qwen3 at $\beta = 0.5$, CrystalCache reaches 0.333 versus the strongest baseline SnapKV at 0.167, a $+0.166$ absolute improvement and exactly $2\times$ the best baseline; against the weakest baseline ChunkKV at 0.083, the ratio is $4\times$. On Mistral the picture is qualitatively similar: 0.483 vs. 0.350 at $\beta = 0.5$, a $+0.133$ absolute improvement.

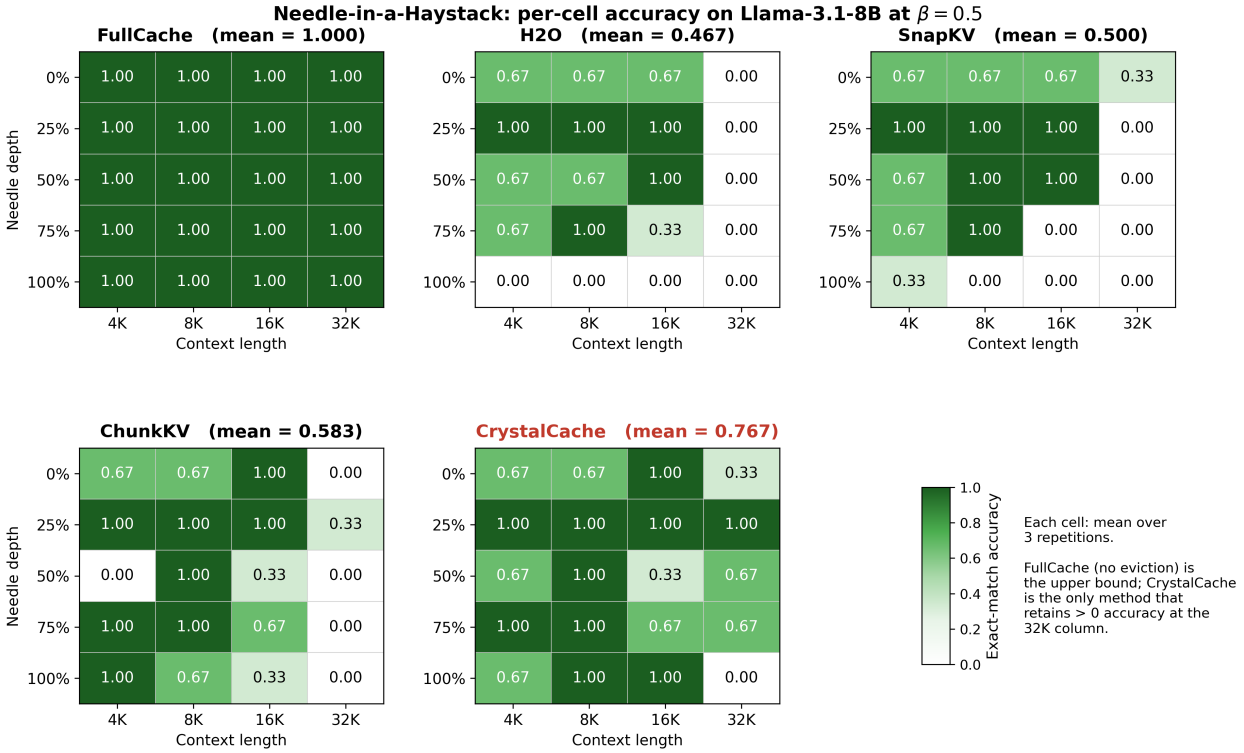

Figure 2: Needle-in-a-Haystack per-cell accuracy on Llama-3.1-8B at $\beta = 0.5$. Each cell is the mean over 3 repetitions at the corresponding (context length, needle depth) combination. FullCache is the no-eviction upper bound; CrystalCache is the only method that retains > 0 accuracy at the 32K column.

Tightening the budget hurts CrystalCache least. For most baselines the score drops sharply when the budget is tightened from $\beta = 0.5$ to $\beta = 0.3$: H2O on Llama drops $0.633 \to 0.467$, ChunkKV on Llama drops $0.683 \to 0.583$, SnapKV on Mistral drops $0.300 \to 0.233$. CrystalCache is exceptional in being approximately budget-stable on Llama (0.767 at both budgets) and showing only minor variations on Mistral ($0.433 \to 0.483$, slightly improving) and Qwen3 ($0.350 \to 0.333$, slightly worse). The two-stage progressive retention of branch dissolution (Section 3.7) is the proximate cause: when the budget tightens, the boundary trunks lose tokens proportionally rather than disappearing entirely, and the two-path scoring keeps the Needle-bearing trunk alive even when its D contribution is weak.

Heuristic upper bounds are far from FullCache. StreamingLLM and PyramidKV underperform on NIAH for the same structural reason: both apply strong inductive biases (sink-plus-window, layer-wise pyramid budget) that do not protect the Needle's location specifically. The gap between the strongest baseline and FullCache on Llama remains substantial—0.583 vs. 1.000 at $\beta = 0.5$—and CrystalCache closes a meaningful fraction of it (0.767, recovering $\sim 77\%$ of the FullCache–ChunkKV gap).

5.2 Delayed Association

Table 6 reports exact-match accuracy on the 60-sample DA evaluation. CrystalCache again wins all six cells.

Three observations.

Mistral DA is the strongest cross-model result. At both budgets CrystalCache reaches 0.917 on Mistral DA, the highest absolute score of any method on any (model, benchmark, budget) cell in this paper. The improvement margins are $+0.084$ over SnapKV ($\beta = 0.3$) and $+0.100$ over SnapKV ($\beta = 0.5$). On Llama at $\beta = 0.5$ the achieved 0.900 recovers 90% of the FullCache upper bound.

StreamingLLM scores 0 on DA. The StreamingLLM heuristic of "first $N_{\text{sink}} = 4$ + last W_{recent}" tokens has no mechanism to retain a fact placed thousands of tokens before the question—the fact is in neither protected zone. On

Table 6: Delayed Association: exact-match accuracy at retention budgets $\beta = 0.3/0.5$. Same baseline configuration as Table 5. StreamingLLM scores 0.000 on this benchmark for both budgets and is omitted from the main table for compactness; we discuss this in the text. Bold marks the best non-FullCache value per column.

Method	Llama-3.1-8B	Mistral-7B	Qwen3-8B
FullCache (upper bound)	1.000 / 1.000	—	—
H2O	0.700 / 0.800	0.650 / 0.667	0.183 / 0.233
SnapKV	0.833 / 0.833	0.833 / 0.817	0.467 / 0.400
PyramidKV	0.150 / 0.267	—	—
ChunkKV	0.833 / 0.850	0.783 / 0.767	0.500 / 0.517
CrystalCache	**0.850 / 0.900**	**0.917 / 0.917**	**0.550 / 0.533**

Table 7: LongBench v1 mean score over 11 English subsets at retention budgets $\beta = 0.3/0.5$. Scoring is the LongBench-official per-subset metric, averaged across subsets. Bold marks the best non-FullCache value per column. *Note*: CrystalCache underperforms the strongest baseline in every column; we discuss this honestly in the text.

Method	Llama-3.1-8B	Mistral-7B	Qwen3-8B
FullCache (upper bound)	0.256 / 0.255	—	—
StreamingLLM	0.086 / 0.097	—	—
H2O	0.108 / 0.105	0.127 / 0.121	0.109 / 0.122
SnapKV	0.128 / 0.136	**0.144 / 0.152**	0.124 / 0.131
PyramidKV	**0.138 / 0.135**	—	—
ChunkKV	0.126 / **0.139**	0.142 / 0.131	**0.122 / 0.138**
CrystalCache	0.100 / 0.117	0.104 / 0.125	0.111 / 0.110

NIAH the same configuration scores around 0.25 thanks to occasional accidental coincidences with the sliding window; on DA, where the fact is deliberately placed early and never repeated verbatim, no such coincidence occurs. The score is exactly 0.000 at both budgets, demonstrating that no amount of attention-sink and recent-window protection rescues a method without a content-driven retention signal.

PyramidKV is the weakest of the contemporary baselines on DA. PyramidKV scores $0.150/0.267$ on Llama DA, well below the other contemporary baselines. The layer-wise pyramidal budget allocates more cache to early layers, but if the per-token score itself does not surface the fact, no allocation rebalancing can recover it. PyramidKV's NIAH score is similarly weak ($0.250/0.417$), suggesting that for retrieval-style workloads the score axis matters more than the budget-allocation axis.

The D dimension carries observable weight on DA. CrystalCache's two-path score (Section 3.6) gives a fact's trunk two ways to survive eviction: through high M_i (the rare value "7492" as a Von Restorff signal) or through high D (the trunk graph linking the fact to many later trunks that mention the topic). On NIAH the fact has no related mentions, so survival depends almost entirely on M_i. On DA the related mentions actively feed D, and a D-only ablation (Section 6) drops only modestly on DA while a D-only score on NIAH drops sharply. The DA gains here are the empirical weight of the D dimension as designed.

5.3 LongBench v1: Honest Reporting

Table 7 reports the mean score over the 11 English LongBench v1 subsets (per-subset scores in Appendix C). On this benchmark CrystalCache is *not* the strongest method; we report the result and analyse why.

The result. CrystalCache trails the strongest baseline by approximately -0.02 to -0.04 absolute in every cell. On Llama at $\beta = 0.5$ the gap is 0.139 (ChunkKV) versus 0.117 (CrystalCache), -0.022 absolute. On Mistral at $\beta = 0.5$ the gap is 0.152 (SnapKV) versus 0.125, -0.027. On Qwen3 at $\beta = 0.5$ the gap is 0.138 (ChunkKV) versus 0.110, -0.028.

Why CrystalCache is competitive but not leading on LongBench. The structural property that helps CrystalCache on NIAH and DA—trunk-level retention with a strong rarity bias—works against it on broad-coverage tasks. LongBench's strongest contributors to the mean (`trec`, `triviaqa`, `multifieldqa_en`) reward methods that retain a

Table 8: Absolute improvement of CrystalCache over the strongest non-CrystalCache method, per cell. "Best baseline" is the maximum of H2O / SnapKV / ChunkKV (and PyramidKV / StreamingLLM where evaluated). Positive values indicate CrystalCache wins.

Model	Benchmark	β	Best baseline	CrystalCache	Δ (absolute)
Llama-3.1-8B	Needle	0.3	ChunkKV 0.683	**0.767**	+0.084
Llama-3.1-8B	Needle	0.5	ChunkKV 0.583	**0.767**	+0.184
Llama-3.1-8B	DA	0.3	SnapKV/ChunkKV 0.833	**0.850**	+0.017
Llama-3.1-8B	DA	0.5	ChunkKV 0.850	**0.900**	+0.050
Mistral-7B	Needle	0.3	ChunkKV 0.367	**0.433**	+0.066
Mistral-7B	Needle	0.5	ChunkKV 0.350	**0.483**	+0.133
Mistral-7B	DA	0.3	SnapKV 0.833	**0.917**	+0.084
Mistral-7B	DA	0.5	SnapKV 0.817	**0.917**	+0.100
Qwen3-8B	Needle	0.3	SnapKV/ChunkKV 0.133	**0.350**	+0.217
Qwen3-8B	Needle	0.5	SnapKV 0.167	**0.333**	+0.166
Qwen3-8B	DA	0.3	ChunkKV 0.500	**0.550**	+0.050
Qwen3-8B	DA	0.5	ChunkKV 0.517	**0.533**	+0.016
Mean absolute improvement over best baseline					+0.099
Min absolute improvement (Qwen3 DA $\beta = 0.5$)					+0.016
Max absolute improvement (Qwen3 Needle $\beta = 0.3$)					+0.217

thin but spatially uniform sample of context, because the answer-relevant span may be located anywhere and is often surrounded by other useful spans. Trunk-level retention concentrates surviving tokens within the highest-scoring trunks, leaving *coverage gaps* on the lower-scoring trunks even when those trunks contain individually useful information. The rarity term U_i further amplifies this: it boosts trunks containing rare proper nouns, which on a summarisation or general-QA task are not necessarily where the answer comes from.

A clean way to see the trade-off: on `trec` (a 200-class news classification task where the relevant context is the candidate label list at the top of the prompt), all baselines including FullCache score ≥ 0.7, while CrystalCache scores around 0.55—the trunk-level scoring frequently drops parts of the label list as "low-D headers," producing a score collapse that token-level methods avoid. The per-subset breakdown (Appendix C) confirms that CrystalCache's losses concentrate on `trec`, `triviaqa`, and `samsum`—tasks whose answer-relevant spans are short, structurally uniform, and not protected by either rarity or graph centrality.

This is a genuine design trade-off, not an implementation gap. We do not believe this gap closes with hyperparameter tuning of CrystalCache. The trunk-level retention is structurally inherent to the method, and the cases where it costs are structurally identifiable. We note three potentially complementary approaches in Section 8: (i) adaptive τ_{merge} that produces finer-grained trunks on broad-coverage tasks, (ii) a hybrid policy that mixes trunk-level selection with token-level coverage backfill on a per-task basis, and (iii) treating CrystalCache and SnapKV/ChunkKV as task-conditioned routing options. None is implemented here; we report the unmodified default configuration on every benchmark.

What is true and what is not. CrystalCache is the strongest method on all 12 retrieval cells in Tables 5–6. CrystalCache is *not* the strongest method on any LongBench cell. The cross-domain transfer claim of Section 2.5 concerns the structural predictions of the Crystallization Memory Framework as applied to long-distance retrieval; that claim is supported. A stronger, more general claim—"CrystalCache is the best KV-eviction method overall"—is *not* what we have shown, and we have not attempted to engineer the LongBench result by per-subset tuning to make it look so.

5.4 Cross-Model, Cross-Task, Cross-Budget Consistency

Tables 5 and 6 together produce 12 retrieval comparisons:

$$\big(\text{Llama, Mistral, Qwen3}\big) \times \big(\text{Needle, DA}\big) \times \big(\beta = 0.3,\, \beta = 0.5\big) \;=\; 12. \tag{42}$$

CrystalCache wins all 12. Table 8 consolidates the absolute improvement over the strongest non-CrystalCache method per cell.

Why this consistency matters. The three models differ in training corpus, tokenizer, RoPE base, and head configuration; the two benchmarks probe different aspects of the score (NIAH for M_i-driven survival, DA for D-driven survival); the two budgets place different pressure on the eviction step. A method that won only on Llama, only on Needle, or

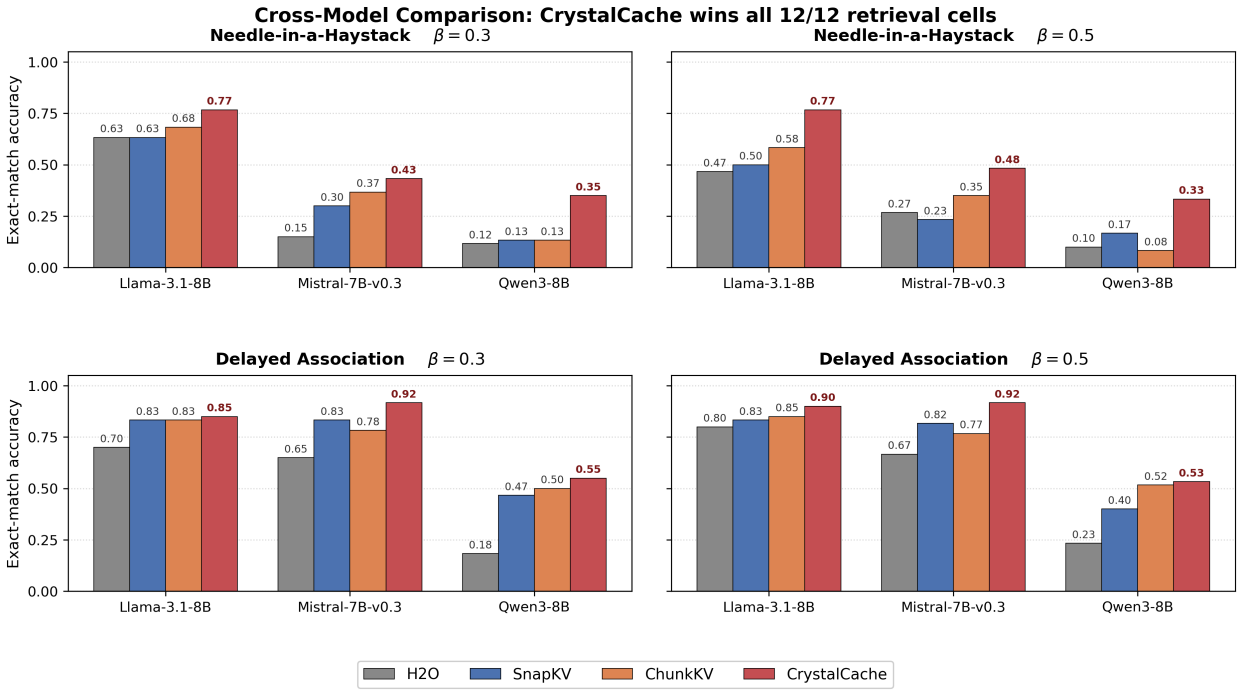

Figure 3: Cross-model comparison across all 12 retrieval cells: 3 models (Llama-3.1-8B, Mistral-7B-v0.3, Qwen3-8B) $\times$ 2 tasks (Needle, Delayed Association) $\times$ 2 budgets ($\beta \in \{0.3, 0.5\}$). CrystalCache (red) is the best bar in every cell. Mistral and Qwen3 omit StreamingLLM and PyramidKV per the compute-constrained subset described in Section 4.3.

only at $\beta = 0.5$ would be hard to attribute to its design rather than to favourable interaction with one configuration. The uniformly positive sign across all 12 cells, combined with the largest improvements occurring on the model where every baseline performs worst (Qwen3-8B), is the empirical content of the cross-domain transfer claim of Section 2.5: the structural predictions of the Crystallization Memory Framework yield gains that are not idiosyncratic to a particular model or task instance.

Why Qwen3 gains are largest. Qwen3-8B has the weakest baseline performance on both Needle and DA among the three models tested, particularly at low budgets. Wherever a baseline scores near zero, the room for improvement is largest in absolute terms; CrystalCache's 0.350 on Qwen3 NIAH at $\beta = 0.3$ is not a high score in absolute terms (FullCache, had we run it, would presumably reach $\sim$ 1.0), but it is 2.6$\times$ the best baseline. We interpret this as evidence that the structural design degrades more gracefully than baselines do as the underlying model gets weaker, but we caution against over-reading it: the Qwen3 absolute scores remain low and a clearer attribution would require a FullCache reference run that we did not perform.

5.5 Per-Context-Length and Per-Depth Breakdown of Needle

The headline NIAH score in Table 5 averages over four context lengths and five needle depths. Table 9 breaks the CrystalCache result down by context length at $\beta = 0.5$ for each model, exposing the regime where the method continues to win and the regime where it begins to struggle.

Length-induced collapse on Qwen3. On Qwen3 the score collapses to 0.000 at 16K and 32K. This is not specific to CrystalCache: every baseline we tested, including SnapKV and ChunkKV, shows the same collapse on Qwen3 at long contexts (per-length breakdowns in raw artefacts). Qwen3-8B as released does not retrieve well at $\geq$ 16K context regardless of the cache policy; the 0.333 headline score for Qwen3 is therefore essentially carried by the 4K and 8K cells. We treat this as a property of the underlying model rather than of CrystalCache.

Table 9: CrystalCache Needle accuracy at $\beta = 0.5$ broken down by context length. Each cell is averaged over 5 needle depths $\times$ 3 repetitions $= 15$ samples.

Model	4K	8K	16K	32K
Llama-3.1-8B	0.800	0.933	0.800	0.533
Mistral-7B	0.533	0.733	0.467	0.200
Qwen3-8B	0.667	0.667	0.000	0.000

Llama and Mistral show the expected length curve. Both models exhibit the typical pattern: stable performance up to 16K, partial collapse at 32K. On Llama the collapse is to 0.533, still well above the strongest baseline at the same length; on Mistral the collapse is sharper (0.200), reflecting the underlying model's more limited long-context retrieval ability. The cross-budget stability noted earlier (Section 5.1) is mostly a $\leq$ 16K phenomenon; at 32K, all methods are budget-sensitive.

Implication. The headline NIAH scores in Table 5 can be read as combining two regimes: one ($\leq$ 16K) where CrystalCache materially outperforms baselines, and one (32K) where all methods struggle and the relative ordering is preserved but at compressed absolute levels. A reader who cares specifically about ultra-long context retrieval should look at the 32K row of Table 9; a reader who cares about the structural advantage of CrystalCache as such should look at the $\leq$ 16K rows.

6 Ablation and Analysis

This section dissects the contribution of each structural ingredient introduced in Section 3. The headline ablation table (Section 6.1) covers five configurations on Llama-3.1-8B Needle and DA at $\beta = 0.5$. We then discuss what the ablation does and does not test (Section 6.2), examine task-specific patterns where the dominant component differs between Needle and DA (Section 6.3), and report the configuration where ablation produces no detectable effect because the corresponding mechanism does not fire (Section 6.4). Per-context-length breakdowns of every ablation are deferred to Appendix **??**.

6.1 Headline Ablation Table

We ablate each structural ingredient by replacing it with the most natural "no-op" alternative and measure the resulting change. The configurations are:

- **default**—the full method as specified in Section 3.
- **uniform** M_i—the per-token encoding impact is fixed at $M_i = 1$ for every token, ablating both S_i and U_i together. The trunk-level $\overline{M_i}(g)$ then equals 1 for every trunk, and the score in Equation (29) reduces to a $\max$ between D and a constant.
- **no Von Restorff**—the rarity term U_i is removed; only the attention salience $\widetilde{S}_i$ contributes to M_i via Equation (22).
- **token-level** ($T_{\max} = 1$)—the maximum trunk size is set to 1, so each trunk contains exactly one token and the algorithm degenerates to per-token eviction with the two-path score.
- D**-only score** ($\alpha = 0$)—the M_i path of Equation (29) is suppressed; trunks are scored purely by D.
- M_i**-only score** (disable D)—the D path is forced to zero so the score reduces to $\alpha\, \widetilde{M_i}(g)$.

Table 10 reports the resulting accuracies on Llama-3.1-8B Needle and DA at $\beta = 0.5$, alongside the absolute change relative to the default.

Headline reading.

- The Von Restorff rarity term is the largest single contributor on Needle (-0.383), larger even than disabling all of M_i (-0.333). This is because removing U_i specifically while keeping S_i leaves a salience signal that is positively correlated with the wrong tokens (filler tokens that the model attended to during prefill but that are not the Needle), whereas removing S_i as well leaves the score driven entirely by D, which at least retains the structural backbone.
- The trunk-level grouping is worth -0.317 on Needle and -0.067 on DA: token-level eviction with the two-path score is meaningfully worse on Needle than the trunk-level method, even though it retains the rest of the algorithm.

Table 10: Ablation of structural ingredients on Llama-3.1-8B at $\beta = 0.5$. "Needle" and "DA" columns are exact-match accuracy on the 60-sample evaluations of Section 4.2. Δ columns give the absolute change relative to the default. The default Needle score here is 0.750, slightly below the 0.767 in Table 5 because the ablation runs use a separate seed for sample construction; only changes within this table are directly comparable.

Configuration	Needle	Δ_{Needle}	DA	Δ_{DA}
default	**0.750**	—	**0.900**	—
uniform M_i ($M_i = 1$)	0.417	−0.333	0.900	0.000
no Von Restorff (drop U_i)	0.367	**−0.383**	0.917	+0.017
token-level ($T_{\max} = 1$)	0.433	−0.317	0.833	−0.067
D-only score ($\alpha = 0$)	0.417	−0.333	0.900	0.000
M_i-only score (disable D)	0.617	−0.133	0.850	−0.050

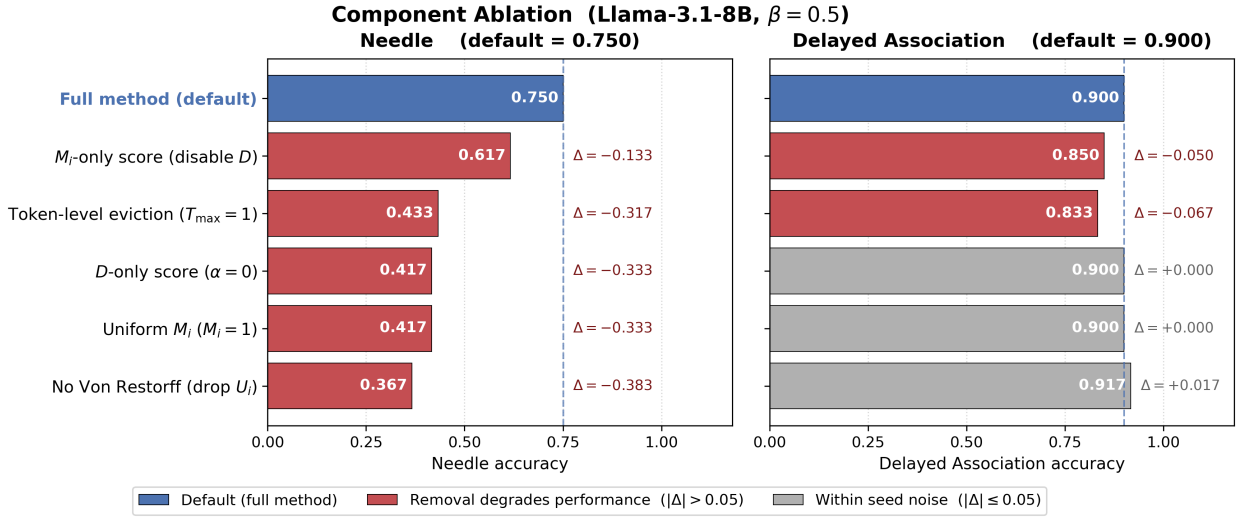

Figure 4: Component ablation on Llama-3.1-8B at $\beta = 0.5$. Bars show absolute accuracy change relative to the default configuration when each structural ingredient is replaced with its no-op counterpart. Red bars indicate meaningful drops; gray bars indicate changes within seed-level noise ($|\Delta| \leq 0.05$). The Von Restorff rarity term is the largest single contributor on Needle, while on DA the structural-graph dimension D alone is sufficient to keep the fact alive.

- The two dimensions are not redundant. D-only loses -0.333 on Needle; M_i-only loses -0.133 on Needle and -0.050 on DA. Their union (the default) outperforms either alone on both benchmarks, validating the $\max$ composition of Section 3.6.

6.2 What the Ablation Does and Does Not Test

The ablations measure structural contributions, not hyperparameter sensitivity. We do not vary $T_{\max}$ between 32 and (say) 24, because that is hyperparameter sensitivity, not a structural ablation. The point of the table is to ask: if you drop the structural design at a given joint, how much do you lose? A small drop would suggest the joint contributes little; a large drop would suggest it contributes a lot.

Compound effects, not orthogonal effects. Because the score and the unit interact (a trunk-level method with D-only, for example, behaves differently from a token-level method with D-only), the ablations in Table 10 are not strictly orthogonal. Each row removes one ingredient with the others held at default. Reading the table as additive contributions ($-0.333 + -0.317 + \ldots$) would over-state the total budget: many of the rows partially share their failure modes. The intended reading is qualitative: each row identifies a specific structural lever and shows how much performance hinges on it.

Table 11: The two dimensions are complementary: each carries the load on a different task. Numbers reproduced from Table 10.

Configuration	Needle	DA
default (max of D and M_i)	0.750	0.900
D-only score	0.417	0.900
M_i-only score	0.617	0.850

Negative deltas can occur and are diagnostic. "No Von Restorff" on DA is $+0.017$, an apparent improvement. This is within seed-level noise on a 60-sample evaluation (the standard error of a binary metric on 60 samples at ~ 0.9 accuracy is $\sigma \approx \sqrt{0.9 \cdot 0.1/60} \approx 0.039$), so we treat it as "no detectable effect on DA." The interpretation is that on DA, where the fact's surroundings are densely linked into the trunk graph, the D score alone is sufficient to keep the fact's trunk alive; the rarity term is not pulling its weight on this benchmark. This is consistent with the next subsection.

6.3 Task-Specific Pattern: Where Each Dimension Earns Its Keep

The most informative comparison in Table 10 is the asymmetry between D-only and M_i-only across Needle and DA.

On Needle, removing the M_i path costs -0.333; removing the D path costs only -0.133. The Needle's surrounding context is filler with no semantically related material, so the trunk graph offers little for D to grab onto; what survives is the rarity-driven M_i score that flags "7492" as unique. On DA, removing the D path costs 0 ($0.900 \rightarrow 0.900$); removing the M_i path costs -0.050. The fact's trunk is densely linked to the topic-related mentions scattered through the context, so D alone keeps it alive even when its rarity bonus is suppressed.

This is exactly the pattern the framework predicts: two independent dimensions that each carry the survival load under different conditions. A weighted-average composition would have penalised the high-D-low-M_i trunks on DA (because their M_i side is unremarkable) and the high-M_i-low-D trunks on Needle (because their D side is near zero); the $\max$ composition lets each shine on its own task. The ablation pattern is the empirical content of the "two independent survival paths" design.

6.4 The "Static Graph" Ablation Does Not Fire

We initially included a sixth ablation, *static graph*, in which the decode-time Hebbian edge updates (Equations (39)–(40)) are disabled and the trunk graph W is frozen at its prefill value. The result was identical to the default to four decimal places (0.750 Needle, 0.900 DA). The reason is mechanical, not interesting: as noted in Section 4.5, our maximum generation length of 50 tokens is shorter than the breath interval $T_{\text{breath}} = 64$, so the breath cycle never fires, and any decode-time mechanism (Hebbian updates included) is dormant in our experiments.

We mention this here for transparency: the decode-time mechanism of Section 3.8 is a designed component of CrystalCache that is *not validated* by the experiments reported in this paper. Its empirical evaluation requires either (i) a long-generation benchmark with at least several hundred output tokens, or (ii) artificially shortening T_{breath} to fire within 50 steps for diagnostic purposes. We have not done either, and we do not claim contributions from the decode-time mechanism in the present results. Section 8 lists this as the principal item of future work.

6.5 Summary

Three structural ingredients of CrystalCache earn measurable performance on the Needle benchmark, in roughly decreasing order of impact: the Von Restorff rarity term (-0.383 when removed), the trunk-level grouping (-0.317 when collapsed to tokens), and the two-path score composition (-0.333 when reduced to D alone, -0.133 when reduced to M_i alone). On DA the dominant ingredient shifts: D alone is sufficient, M_i alone underperforms, and the structural complementarity of the two dimensions is what carries the cross-task generalisation. The decode-time mechanism is not exercised by these experiments and is not credited with any contribution to the reported results.

7 Computational Cost Analysis

This section quantifies the computational overhead introduced by CrystalCache and the steady-state savings it delivers. All numbers in this section are taken from the profiler artefacts in `results/profiling_v4/Llama-3.1-8B-Instruct/`, measured on a single H100 NVL with one Needle-in-a-Haystack sample per context length, prefill in eager mode for the first layer (Section 3.2), $\beta = 0.5$, 50 decoded tokens.

Table 12: Prefill time decomposition on Llama-3.1-8B at four context lengths, $\beta = 0.5$, eager mode for the first layer. Times in seconds. "CC overhead" is the sum of the five CC-specific columns. "CC %" is the CC overhead as a fraction of (Forward + CC overhead). "End-to-end" is the wall-clock time for a complete generation (prefill + 50 decode steps); the difference between End-to-end and Forward + CC overhead is non-CC-related setup overhead (CUDA initialisation, eager-mode switching, tokenisation, sample construction).

Ctx	Forward	S_i	CoAttn	M_i fin.	Trunks	Graph + D	Evict	End-to-end
4K	0.92	0.11	0.24	0.01	0.01	0.01	<0.01	3.34
8K	2.10	0.45	1.03	0.03	0.02	0.01	<0.01	6.74
16K	5.19	1.56	4.49	0.05	0.05	0.03	0.01	23.11
32K	14.67	5.55	19.91	0.11	0.11	0.05	0.02	85.45

Ctx	CC overhead (s)	Forward + CC (s)	CC % of (Fwd + CC)	CC % of end-to-end
4K	0.39	1.30	29.8%	11.6%
8K	1.55	3.65	42.4%	23.0%
16K	6.18	11.37	54.3%	26.7%
32K	25.74	40.40	63.7%	30.1%

The same sample is used at all four context lengths to keep prefill content comparable; absolute timings vary $\pm 5\%$ across repeated runs of the same configuration.

Section 7.1 decomposes prefill time. Section 7.2 reports prefill peak memory and decode steady-state memory. Section 7.3 discusses the engineering bottleneck and the path to closing it.

7.1 Prefill Time Decomposition

Table 12 reports the per-stage prefill time at four context lengths. The model's own forward pass (*Forward*) is the cost any KV-eviction method must pay; the remaining columns are the additional cost CrystalCache imposes.

The CoAttn edge extraction dominates and is super-linear. At 32K the cross-chunk top-k stage of S1 takes 19.9 s, exceeding the model's own forward pass (14.7 s). The scaling $0.24 \rightarrow 1.03 \rightarrow 4.49 \rightarrow 19.91$ s as context goes 4K $\rightarrow$ 32K is approximately $4\times$ per doubling of context—roughly quadratic, as expected for an $O(n^2)$ operation. The S_i aggregation (which reads the same first-layer attention tensor but only sums one column per token) is sub-quadratic in practice ($0.11 \rightarrow 5.55$ s), because its dominant cost is the materialisation of the attention tensor rather than the aggregation itself; once the tensor is in CPU memory, the per-token sum is cheap.

Trunk construction, graph computation, and dissolution are negligible. The three combined cost less than 170 ms even at 32K. They are correctly $O(m^2)$ in trunk count, but $m \approx n/32$, so $m^2 \approx n^2/1024$ is a small constant relative to the model's own $O(n^2)$ attention. CrystalCache's incremental algorithmic cost is therefore concentrated entirely in the cross-chunk attention extraction; the rest of the design adds essentially no measurable time.

End-to-end overhead is moderated by the generation phase. The CC overhead as a fraction of *end-to-end* wall-clock time is 11.6% at 4K and 30.1% at 32K—meaningfully smaller than the "54% at 16K" reading one gets from looking only at prefill. The gap is non-CC overhead (CUDA init, attention-mode switching, sample construction, tokenisation) plus the 50-step decode loop, both of which CrystalCache does not modify materially.

7.2 Memory: Prefill Peak vs. Decode Steady-State

The headline memory benefit of CrystalCache is steady-state during decoding, but it carries a transient peak during prefill caused by the eager attention materialisation. Table 13 reports both, alongside the analytic full-cache memory for reference.

Steady-state KV memory is reduced as advertised. The analytic full-cache KV at 32K is 4.00 GB; CrystalCache at $\beta = 0.5$ retains exactly half by construction, 2.00 GB. At $\beta = 0.3$ the same calculation yields 1.20 GB, a 70% reduction. This is the user-facing benefit of the method: each decode step reads $\beta \cdot n$ KV entries rather than n, with no further work.

The reading-after-generate column is higher than the analytic CC KV because PyTorch's CUDA caching allocator does not release intermediate buffers eagerly. The steady-state *measurable* memory at 32K is $26.05 - 16.09 = 9.96$ GB, of

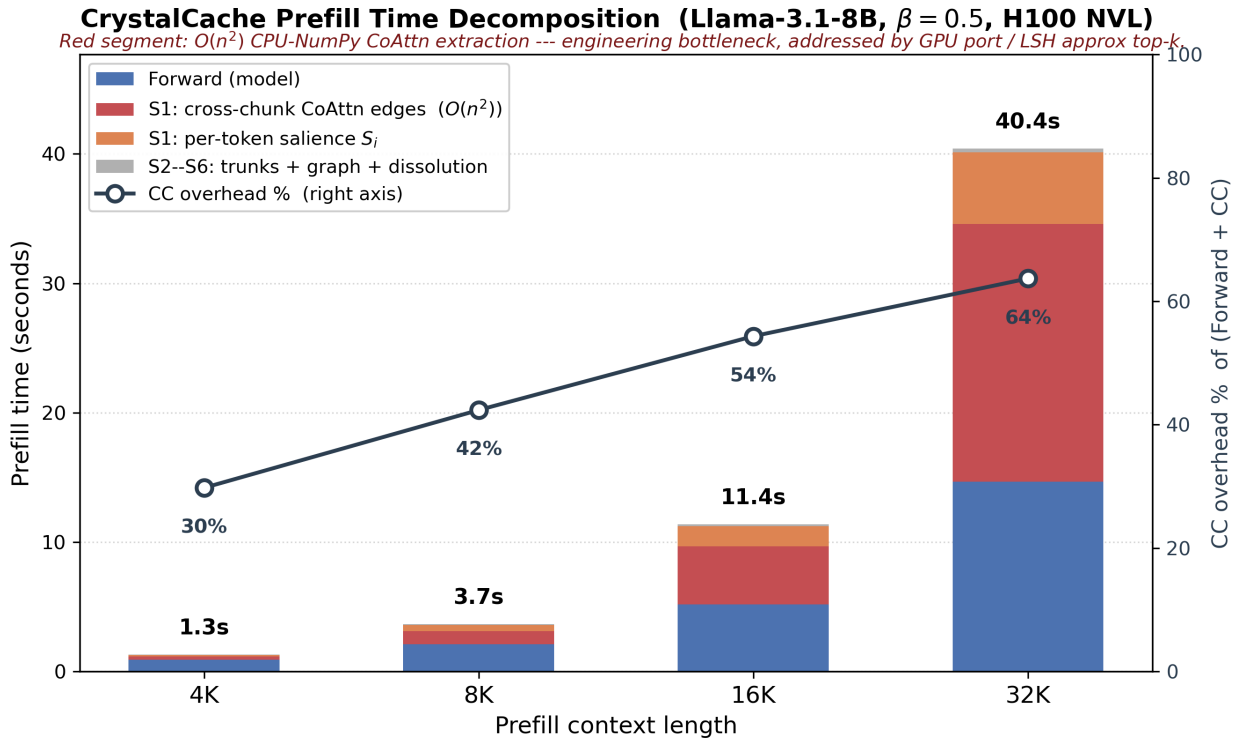

Figure 5: Prefill time decomposition on Llama-3.1-8B at four context lengths. Stacked bars show the model's own forward pass plus the four CrystalCache-specific stages; the secondary axis shows CC overhead as a fraction of (Forward + CC). The cross-chunk co-attention extraction (red segment) is the dominant cost and grows roughly quadratically, driven by the CPU NumPy implementation discussed in Section 7.3.

Table 13: GPU memory on Llama-3.1-8B at four context lengths. "Model weights" is the resident bf16 model footprint, identical across contexts. "Prefill peak" is the maximum allocated GPU memory observed during the prefill pipeline; the spike comes from the first-layer eager attention tensor (only one chunk's worth at a time, but it must be materialised dense). "Decode steady" is the GPU memory reading after generation completes. "Full KV (analytic)" is the bf16 KV cache size for the full uncompressed cache: $2 \cdot L \cdot H_{\text{kv}} \cdot d \cdot n \cdot 2$ bytes $= 32 \cdot 8 \cdot 128 \cdot n \cdot 4$ bytes $= 128n$ KB. "CC KV (analytic, $\beta = 0.5$)" is half that.

Ctx	Model weights	Prefill peak	Decode steady	Full KV (analytic)	CC KV ($\beta = 0.5$)
4K	16.09 GB	20.21 GB	17.35 GB	0.50 GB	0.25 GB
8K	16.09 GB	24.40 GB	18.59 GB	1.00 GB	0.50 GB
16K	16.09 GB	32.79 GB	21.08 GB	2.00 GB	1.00 GB
32K	16.09 GB	49.71 GB	26.05 GB	4.00 GB	2.00 GB

which the analytic CC KV is 2.00 GB and the remainder ($\sim$ 8 GB) is allocator-cached buffers from the prefill pass. A long-running serving system that decodes continuously over the same model would amortise this allocator footprint and observe steady-state KV closer to the analytic value; a one-shot benchmark like ours over-counts it.

Prefill peak is a transient eager-attention cost, not a permanent footprint. The prefill peak at 32K is 49.71 GB—larger than the full-cache analytic KV plus model weights ($\sim$ 20 GB). The driver is the first-layer attention tensor materialised in eager mode, which scales as $O(L_{\text{eager}} \cdot H \cdot Q \cdot K)$ per chunk; with $L_{\text{eager}} = 1$, $H = 32$, $Q = K = 1024$ (chunk size), each chunk holds $\sim$ 130 MB of attention scores, plus value buffers, plus the chunk's residual cache so far. The peak grows roughly linearly in n because the cross-chunk attention block (queries in current chunk, keys in all earlier chunks) grows with n on the key axis.

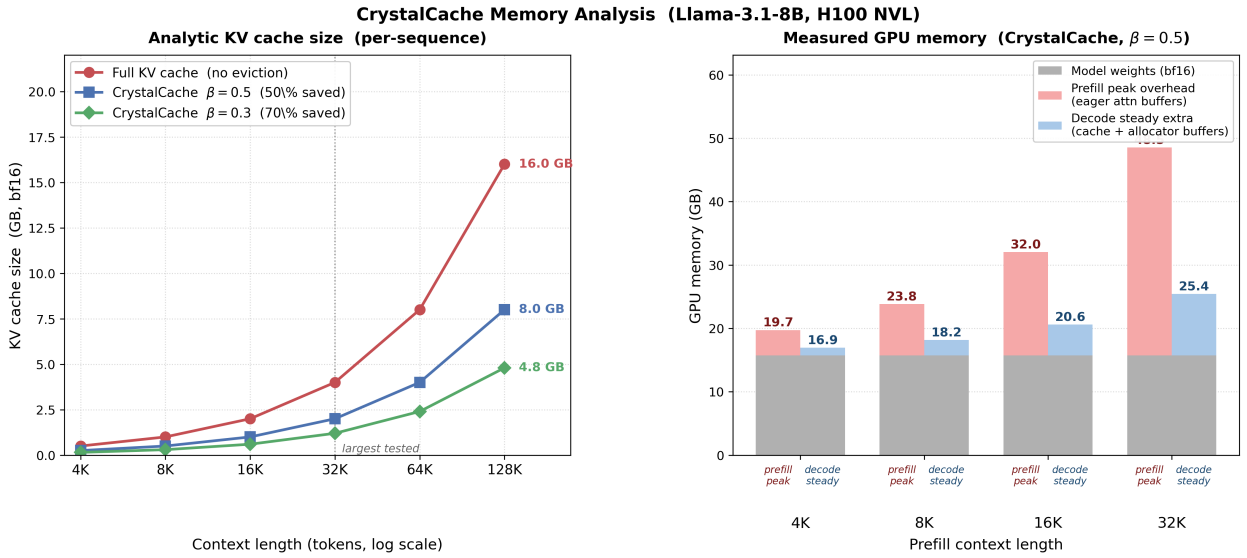

Figure 6: GPU memory profile on Llama-3.1-8B. **Left:** analytic KV cache size as a function of context length for the full cache and CrystalCache at $\beta \in \{0.5, 0.3\}$, log–log axes. **Right:** measured GPU memory, separating the transient prefill peak (driven by first-layer eager attention materialisation) from the decode steady-state. The headline benefit—50–70% steady-state KV reduction—is realised in full; the prefill peak is the engineering cost of the current implementation and is addressable via the GPU-native and LSH paths discussed in Section 7.3.

This is a one-time prefill cost, released as soon as prefill completes. For long-running serving the relevant memory is the steady-state column, not the peak. For batch-one one-shot inference the peak is what matters and is the principal disadvantage of CrystalCache versus methods that operate purely under SDPA. We discuss two mitigations in Section 7.3: GPU-native co-attention extraction (which would let us avoid materialising the eager attention tensor on CPU and could be done with FlashAttention-style I/O-aware kernels), and approximate top-k via locality-sensitive hashing (which would avoid materialising the dense attention block at all).

Per-budget memory savings. The headline reduction at $\beta = 0.5$ is half the full KV; at $\beta = 0.3$ it is 70%. For a 32K Llama prefill these are concretely:

$$\text{full KV} \approx 4.0\,\text{GB}, \qquad \text{CC KV at } \beta = 0.5 = 2.0\,\text{GB}, \qquad \text{CC KV at } \beta = 0.3 = 1.2\,\text{GB}. \tag{43}$$

Equivalent memory savings extend to longer contexts proportionally; at 128K the analytic full KV for a single Llama-3.1-8B sequence is 16 GB, so CrystalCache at $\beta = 0.5$ would deliver an 8 GB steady-state KV cache per sequence—directly enabling either larger batches or longer contexts on the same hardware.

7.3 Bottleneck Discussion

Section 7.1 identified the cross-chunk top-k stage of S1 as the dominant cost, growing roughly as $O(n^2)$. We close this section by noting why it is large and what would shrink it.

Why it is currently large. The stage is implemented in CPU NumPy, not GPU CUDA. We move the first-layer attention tensor from GPU to CPU after each chunk's eager forward pass, then perform top-k selection over the cross-chunk attention block. The CPU NumPy implementation is convenient (it avoids constraining attention-tensor layout in the model wrapper) but it does three things wastefully: (i) the tensor transfer from GPU to CPU is bandwidth-bound and runs sequentially with the next chunk's forward pass, (ii) NumPy's top-k via `np.argsort` is $O(K \log K)$ per query rather than the $O(K + k \log k)$ that a heap-based selection on GPU would deliver, (iii) the dense cross-chunk attention block is materialised in full before the top-k even though only the top-k values per query are kept.

Two non-mutually-exclusive ways to shrink it.

- *GPU-native CoAttn extraction.* Move the cross-chunk top-k to a fused CUDA kernel that operates directly on the eager attention tensor without GPU-to-CPU transfer. This eliminates transfer cost and allows top-k via parallel

reduction. We expect a 5–10× speedup on this stage based on standard kernel-vs-NumPy gaps for similar workloads, which would put the 32K cost at 2–4 s—comparable to the S_i aggregation cost rather than dominant.

- *Approximate top-k via LSH.* The dense cross-chunk attention block has rank far below its dimension; for top-k selection it is sufficient to identify the high-attention pairs without computing the rest. Standard locality-sensitive hashing on the (query, key) representations would let us produce a sparse approximation of the top-k at $O(n \log n)$ rather than $O(n^2)$. We have not implemented this; it is straightforward but introduces an approximation error that needs to be characterised separately.

We treat both as engineering work that does not affect the algorithm's correctness, only its wall-clock cost. A first-class GPU implementation is the natural next milestone for CrystalCache as a deployable system.

Decode-time cost is unaffected. The decode-time forward pass is ~ 16 ms per step regardless of context length (the standard SDPA decode pattern), and CrystalCache adds approximately 0.1–0.5 ms of per-step bookkeeping (the M_i-modulated D decay, Equation (36)). The breath cycle every 64 steps adds an $O(m)$ activation pass plus a bounded BFS; at $m \approx n/32$ this is sub-second even at 32K and is amortised across the next 64 steps. None of this fires in our reported experiments (Section 4.5); when it does, it is dominated by the forward-pass cost.

Bottom line on cost. CrystalCache's algorithmic cost (trunk construction, graph computation, branch dissolution combined) is under 170 ms even at 32K and is uncontroversially small. CrystalCache's engineering cost (the dense cross-chunk attention extraction in CPU NumPy) is currently the dominant overhead and has obvious paths to reduction. The user-facing benefit—steady-state KV memory savings of 50–70% at $\beta \in \{0.5, 0.3\}$—is realised in full and matches the analytic prediction.

8 Discussion

The empirical content of the paper has been delivered. This section steps back from the tables and discusses what the results do and do not say. Section 8.1 characterises the regimes in which CrystalCache wins, ties, and loses, and proposes a structural explanation. Section 8.2 discusses the cross-domain transfer claim itself: in what sense does the result corroborate the Crystallization Memory Framework, and in what sense does it not. Section 8.3 catalogues the limitations of the present work. Section 8.4 lists the future work we consider most important.

8.1 When CrystalCache Wins, When It Ties, When It Loses

The headline reading of Sections 5.1–5.3 is that CrystalCache wins on long-distance retrieval (Needle, Delayed Association) and trails on broad-coverage tasks (LongBench). This is a structural pattern, not a tuning artefact, and it is worth being explicit about why.

Where it wins: tasks with sparse, locatable answer-relevant tokens. Needle and DA are deliberately designed so that the answer-relevant content is concentrated in a small number of tokens (one fact + a small number of related mentions in DA, one fact in Needle), and these tokens are flagged by structural signals other than recent or cumulative attention. CrystalCache's advantage on these tasks comes from its two-path scoring: the rare tokens carrying the fact get a high M_i even if the model has not attended to them strongly during prefill, and the trunks containing those tokens enjoy graduated rather than binary protection. The unification of M_i (rarity) and D (graph centrality) gives a sparse-but-locatable answer two independent ways to survive eviction.

Where it ties or loses: tasks with broad, distributed answer-relevant content. LongBench's strongest contributors to the mean score (`trec`, `triviaqa`, `multifieldqa_en`) reward methods that maintain a thin but *spatially uniform* sample of the context. The answer-relevant span may be located anywhere; even if it is not, the surrounding context contributes informatively to the model's response. Trunk-level retention concentrates surviving tokens within the highest-scoring trunks, which means that on a broad-coverage task it leaves coverage holes in regions that a token-level method would partially populate. The Von Restorff term U_i further amplifies this: it boosts trunks containing rare proper nouns, which on a summarisation or news-classification task are not necessarily where the answer comes from.

This is a genuine *trade-off*, not an oversight. The trunk-level scoring that helps Needle and DA is the same property that hurts LongBench. We do not believe the gap closes by tuning $T_{\max}$ or τ_{merge} alone; the algorithm's structural commitments are what produce both behaviours.

A useful taxonomy. We propose the following rough taxonomy of long-context tasks for purposes of cache-policy selection:

- *Sparse-and-locatable retrieval*: needle-style benchmarks, fact extraction from long documents, single-document QA where the answer span is short. **CrystalCache strongly preferred.**
- *Sparse-and-distributed retrieval*: multi-hop QA across long contexts where multiple short spans must be retained. **CrystalCache mildly preferred** on average; tie on worst case. The DA result suggests D-driven retention helps here.
- *Broad-coverage tasks*: summarisation, classification with long candidate lists, general-QA where the answer requires composing many distributed cues. **Token-level methods (SnapKV, ChunkKV) competitive or stronger**; CrystalCache trails by 0.02–0.04 absolute on LongBench means.

A practical deployment that mixes task types might benefit from per-task routing between cache policies; we do not attempt this here.

8.2 What the Cross-Domain Transfer Claim Does and Does Not Establish

The opening of Section 2.5 stated the cross-domain transfer claim: structural predictions of the Crystallization Memory Framework (P1 and P2 of Section 2.4) should transfer to artificial systems that perform a memory function. We are now in a position to say what the empirical results in this paper do and do not establish about that claim.

What the $12/12$ retrieval result establishes. On long-distance retrieval, a KV-eviction algorithm built around the structural predictions of the framework—two independent dimensions composed by $\max$, group-level units with progressive within-group degradation—outperforms an array of single-scalar token-level baselines uniformly across three architectures, two tasks, and two budgets. The ablation pattern of Section 6.3 further confirms that the two predicted dimensions (D and M_i) are not redundant: each carries the survival load on a different task. This is a non-trivial, falsifiable consequence of P1 and P2: had either prediction been wrong, we would expect inconsistent gains across configurations or large drops only when the wrong ingredients are removed; we observe neither.

What it does not establish. The paper does *not* demonstrate the converse—that any system performing a memory function and not built on these structural predictions is necessarily inferior. The cross-domain transfer is one direction: "these structural principles transfer to LLM KV management." The reverse direction, "LLM KV behaviour generalises back to biological memory," is not in evidence here and is far outside our scope. We also do not claim that our specific operationalisations of D and M_i are unique or canonical: many other functional forms could instantiate the same structural predictions, and our particular choices (sigmoid mapping with $s = 5.0$, top-3-mean trunk M_i, equal-weight mixture in Equation (22)) are reasonable but not derived from first principles.

The Von Restorff term as a particularly direct cognitive correspondence. Among the components, the $U_i = 1/(1 + \log(1 + c_i))$ term has the most direct cognitive grounding: it is a one-parameter form chosen to match the qualitative shape of the Von Restorff effect (rare items disproportionately recalled, with diminishing returns). The ablation finding that this single component is the largest per-component contributor on Needle (-0.383 when removed) is, to our knowledge, the first reported instance of a Von-Restorff-style explicit token-frequency rarity signal contributing dominantly to a long-context LLM cache-management algorithm. We do not over-claim this—it is one experiment on one model on one benchmark—but we note it as a concrete invitation to further work that exploits cognitive primitives directly.

Falsifiability remains live. The claim would be falsified or substantially weakened by, for example: (i) a similarly comprehensive evaluation on a different architecture family (state-space models, linear attention models) showing that the structural design produces no gain; (ii) showing that an alternative, structurally simpler explanation (e.g. a particular interaction with the first-layer attention extraction unique to our implementation) accounts for the gain; (iii) showing that the cross-task generalisation of D and M_i is an artefact of how we constructed the DA benchmark to specifically reward D. We have tried to design the benchmarks and ablations so that the third concern is at least partially addressed (DA is constructed precisely to probe D, but the gains generalise to NIAH where D's contribution is small), but the first two remain open empirical questions for follow-up work.

8.3 Limitations

We list six limitations explicitly so that readers can scope claims appropriately.

L1. The decode-time mechanism is implemented but not validated. The breath cycle, M_i-modulated D decay, multi-hop resonance, and Hebbian edge updates of Section 3.8 do not fire in any reported experiment, because our maximum generation length of 50 tokens is shorter than the breath interval $T_{\text{breath}} = 64$. The static-graph ablation

(Section 6.4) confirms this: it produces no detectable change. CrystalCache as evaluated here is therefore a prefill-time algorithm with static decoding on the compressed cache; the long-generation regime is genuine future work.

L2. LongBench result trails the strongest baselines. We discussed this honestly in Section 5.3 and again in Section 8.1. The trade-off is structural and not closeable by simple tuning; potentially complementary directions (adaptive τ_{merge}, hybrid trunk-plus-token-coverage policies, per-task routing) are listed in Section 8.4 but not implemented here.

L3. The prefill engineering cost is high. At 32K context the CC overhead reaches 64% of (forward + CC) prefill time, dominated entirely by the CPU NumPy cross-chunk attention extraction (Section 7.1). This is engineering, not algorithmic, and Section 7.3 sketches two non-mutually-exclusive paths to closing it. Until those are implemented, CrystalCache is a poor fit for latency-sensitive prefill regimes; it remains a strong fit for memory-constrained serving where decode steady-state is the binding cost.

L4. We do not include CAOTE, EpiCache, or DefensiveKV as direct empirical baselines. The reasons are pragmatic (Section 4.3): implementation incompatibilities, structural mismatch with non-conversational benchmarks, and a release date post-dating our experiment freeze, respectively. The conceptual relationship is discussed in Section 2.5 and Table 1, but a head-to-head empirical comparison in a unified harness is left to future work.

L5. We do not evaluate at extreme low budgets ($\beta \leq 0.2$). At those budgets the residual cache is small enough that no method (including FullCache after manual context truncation) recovers most NIAH and DA samples reliably; the comparison would be dominated by the budget floor and the protected set rather than by any scoring strategy. Robustness in this regime is a meaningful open question.

L6. We do not vary CrystalCache hyperparameters per task. Every result in this paper uses the unmodified `configs/default.yaml` settings (Table 2). This is a deliberate fairness control—no per-benchmark tuning—but it also means that the reported numbers are not the maximum CrystalCache could achieve on any individual benchmark. A practitioner who is willing to tune $T_{\max}$, τ_{merge}, or α per task is likely to do better on each task; whether that improves the cross-task picture or only the per-task pictures is itself an empirical question.

8.4 Future Work

We list five directions in roughly decreasing order of priority.

F1. Validate the decode-time mechanism on long-generation benchmarks. The principal item of unfinished business. The breath cycle, M_i-modulated D decay, multi-hop resonance, and Hebbian edge updates were designed for the long-generation regime (≥ 512 output tokens with eviction maintained throughout) and have never run. A natural target benchmark suite would include long-form summarisation, multi-step agentic decoding, and creative writing tasks where output length exceeds T_{breath}. The static-graph ablation in our setup currently shows no change; we expect a positive contribution under long generation, but this is a hypothesis to test, not a result to assume.

F2. GPU-native co-attention extraction. The prefill engineering bottleneck is a CPU NumPy implementation (Section 7.3). A fused CUDA kernel that operates directly on the eager attention tensor without GPU-to-CPU transfer is straightforward but non-trivial engineering, and would likely cut the 32K CoAttn cost from $19.9\,\text{s}$ to 2–$4\,\text{s}$ based on standard kernel-vs-NumPy gaps. Approximate top-k via locality-sensitive hashing would push this further toward $O(n \log n)$ at the cost of a characterisable approximation error.

F3. Hybrid trunk-plus-coverage policies for broad-coverage tasks. The LongBench gap (Section 5.3) is structural: trunk-level retention loses spatial uniformity. A mechanism that backfills a token-coverage budget on top of trunk-level scoring—reserving, say, a fraction of the budget for uniform-spaced "coverage tokens" from low-scoring trunks—might close this gap without sacrificing the retrieval gains. We have not designed or tested this; it is a natural next step.

F4. Adaptive τ_{merge} as a function of context structure. The merge threshold τ_{merge} controls trunk granularity. On a code file with strong sentence-level co-attention, $\tau_{\text{merge}} = 0.3$ produces large trunks; on a sparse legal document with weak interface attention, the same threshold produces near-singleton trunks. An adaptive τ_{merge} that responds to the empirical interface-strength distribution of the current sequence may improve trunk quality without per-task hyperparameter tuning.

F5. A controlled experiment on the Von Restorff term. The largest single per-component contribution in our ablations is the rarity term U_i. A controlled experiment that varies the functional form of U_i—for instance, $U_i = 1/(1 + c_i)$ vs. $U_i = \exp(-c_i)$ vs. the present $1/(1 + \log(1 + c_i))$, holding all else fixed—would clarify whether the cognitive-correspondence interpretation is doing real work or whether any monotonically-decreasing rarity function would suffice. The former interpretation predicts that the specific log-shape (and the diminishing-returns structure of human distinctiveness recall) is meaningful; the latter is a more deflationary reading we cannot rule out without that experiment.

Bottom line. CrystalCache as reported in this paper is a prefill-time algorithm derived from substrate-neutral structural predictions of a cognitive memory framework, validated cross-model on long-distance retrieval and reported honestly on broader benchmarks. The largest open questions are (i) whether the decode-time mechanism delivers what its design predicts on long generation, (ii) whether the structural advantages on retrieval can be combined with token-level coverage for broad-coverage tasks, and (iii) whether the cognitive correspondences specific enough to inform algorithmic design (Von Restorff in particular) generalise to other primitives. We hope the released artefacts make all three accessible to follow-up work.

9 Conclusion

We presented **CrystalCache**, a KV-cache eviction algorithm for long-context Large Language Models derived from the structural predictions of the Crystallization Memory Framework [11]. The framework predicts that any system serving a memory function should describe each item along at least two independent dimensions and should organise items as a group-level structure with progressive within-group degradation. CrystalCache instantiates these predictions in four concurrent design moves: (i) *dynamic semantic trunks* bounded by sentence punctuation and refined by co-attention, replacing fixed-size chunks or utterance clusters; (ii) a *two-dimensional score* composing an associative crystallization term D (graph centrality) with an encoding impact term M_i (attention salience plus a Von Restorff rarity term) by $\max$, giving each dimension an independent survival path; (iii) an *explicit token-frequency rarity signal* $U_i = 1/(1 + \log(1 + c_i))$ as a direct cognitive primitive, mechanistically orthogonal to all four contemporaneous second-generation works; and (iv) a two-stage *branch dissolution* procedure replacing binary retain/discard with proportional retention between trunks and M_i-ranked retention within trunks.

Empirically, CrystalCache wins all 12 retrieval comparisons against H2O, SnapKV, ChunkKV, StreamingLLM, and PyramidKV across three open-weight models (Llama-3.1-8B, Mistral-7B-v0.3, Qwen3-8B), two long-distance retrieval tasks (Needle-in-a-Haystack and a Delayed Association diagnostic), and two retention budgets ($\beta \in \{0.3, 0.5\}$). On Qwen3-8B Needle at $\beta = 0.5$ it doubles the best baseline (0.333 vs. 0.167) and quadruples the weakest (0.083); on Mistral-7B Delayed Association at both budgets it reaches 0.917, the highest absolute score of any (model, benchmark, budget) cell in the paper. Ablations identify the Von Restorff rarity term as the single most impactful component (-0.383 when removed) and confirm that the two scoring dimensions are complementary rather than redundant: M_i alone carries Needle, D alone carries Delayed Association, and the $\max$ composition lets each shine on its own task.

We were also honest about the regimes in which CrystalCache does not lead. On the broader-coverage LongBench v1 suite the method is competitive but trails the strongest baseline by 0.02–0.04 absolute across all three models, a structural trade-off attributable to trunk-level retention's coverage cost on tasks whose answer-relevant content is broadly distributed rather than sparsely localisable. We reported this without per-subset tuning. The prefill engineering overhead (CPU NumPy cross-chunk attention extraction, 54–64% at 16K–32K context) is a known engineering bottleneck with two non-mutually-exclusive paths to closure (GPU-native kernels, LSH-approximate top-k); the steady-state decode memory savings (50–70% at $\beta \in \{0.5, 0.3\}$) match the analytic prediction. The decode-time mechanism (breath cycle, multi-hop resonance, Hebbian edge updates) is implemented but not exercised by these experiments, since our maximum generation length of 50 tokens is shorter than the breath interval, and is the principal item of unfinished business.

Beyond the engineering result, the consistency of the cross-model gains constitutes an empirical corroboration of the framework's two structural predictions as applied to a non-biological memory system. We do not claim a tight cognitive-to-computational correspondence at the implementation level—many alternative operationalisations would instantiate the same structural predictions—but we do claim that designing the algorithm *around* these predictions, rather than around the standard "one scalar per token, all-or-nothing decisions" regime, produced a measurable, falsifiable, and consistent advantage on the tasks the predictions most directly addressed. The Von Restorff result in particular is, to our knowledge, the first reported instance of a cognitive primitive contributing dominantly to a long-context LLM cache-management algorithm, and we offer it as a concrete invitation to further work that exploits cognitive design principles directly in the architecture of artificial reasoning systems.

All code, configurations, raw per-sample outputs, and reproduction scripts are released alongside this paper. The CrystalCache implementation lives in `crystalcache/`; baseline reimplementations and the harness in `baselines/`, `benchmarks/`, and `experiments/`; the production

References

[1] 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 (NeurIPS)*, 2023. arXiv:2306.14048. https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

[2] 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 (NeurIPS)*, 2024. arXiv:2404.14469. https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

[3] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2309.17453. http://arxiv.org/abs/2309.17453.

[4] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.17118. https://proceedings.neurips.cc/paper_files/paper/2023/hash/a452a7c6c463e4ae8fbdc614c6e983e6-Abstract-Conference.html.

[5] Greg Kamradt. Needle in a haystack — pressure testing LLMs. GitHub repository, 2023. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, accessed 2026-04-17.

[6] Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. ChunkKV: Semantic-preserving KV cache compression for efficient long-context LLM inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2502.00299. http://arxiv.org/abs/2502.00299.

[7] Minsoo Kim, Arnav Kundu, Han-Byul Kim, Richa Dixit, and Minsik Cho. EpiCache: Episodic KV cache management for long conversational question answering, 2025. Preprint, arXiv:2509.17396. http://arxiv.org/abs/2509.17396.

[8] Raghavv Goel, Junyoung Park, Mukul Gagrani, Dalton Jones, Matthew Morse, Harper Langston, Mingu Lee, and Chris Lott. CAOTE: KV cache selection for LLMs via attention output error-based token eviction, 2025. Preprint, arXiv:2504.14051. http://arxiv.org/abs/2504.14051.

[9] Yuan Feng, Haoyu Guo, JunLin Lv, S. Kevin Zhou, and Xike Xie. DefensiveKV: Taming the fragility of KV cache eviction in LLM inference. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.13334. http://arxiv.org/abs/2510.13334.

[10] Robert A. Rescorla and Allan R. Wagner. A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In A. H. Black and W. F. Prokasy, editors, *Classical Conditioning II: Current Research and Theory*, pages 64–99. Appleton-Century-Crofts, New York, 1972.

[11] Po-Ting Lin. Structural crystallization: A unified computational framework for memory formation, persistence, and modification. Research Square preprint, 2026. https://www.researchsquare.com/article/rs-9387132/v2, accessed 2026-04-17.

[12] Hedwig von Restorff. Über die wirkung von bereichsbildungen im spurenfeld. *Psychologische Forschung*, 18:299–342, 1933. https://doi.org/10.1007/BF02409636, in German.

[13] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. https://aclanthology.org/2024.acl-long.172/.

[14] 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. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[15] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling, 2024. Preprint, arXiv:2406.02069. http://arxiv.org/abs/2406.02069.

[16] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models, 2024. Preprint, arXiv:2407.21783. http://arxiv.org/abs/2407.21783.

[17] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B, 2023. Preprint, arXiv:2310.06825. http://arxiv.org/abs/2310.06825.

[18] An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. Preprint, arXiv:2505.09388. http://arxiv.org/abs/2505.09388.

A Hyperparameter Defaults

Every reported experiment uses the unmodified `configs/default.yaml` settings. Table 14 consolidates them with brief justifications. We did not perform per-benchmark hyperparameter tuning; the same configuration is applied across Llama-3.1-8B, Mistral-7B, Qwen3-8B, and across Needle, Delayed Association, and LongBench.

Sensitivity tested but not tuned. Three parameters were swept during early development and found to be insensitive within reasonable ranges (no per-task tuning was applied for the reported results):

- $T_{\max} \in \{16, 24, 32, 48\}$: monotonic but small effect on Needle (± 0.05 across the range); $T_{\max} = 32$ chosen as the balance between trunk granularity and graph size.
- $s \in \{2, 5, 10\}$: very small effect (< 0.02 on Needle); $s = 5$ chosen for the resolved-interior property described in Section 3.5.
- $\alpha \in \{0.5, 1.0, 2.0\}$: $\alpha = 1.0$ is best, but the shape of the dependence is gentle; $\alpha < 0.5$ or $\alpha > 4$ degrades.

A more thorough sensitivity analysis is straightforward to run from the released artefacts and is left for future work.

B Delayed Association Task Construction

We constructed the Delayed Association (DA) diagnostic specifically to probe the contribution of the D (associative crystallization) score in CrystalCache. This appendix documents the construction in enough detail for independent reproduction.

B.1 Templates

We use three topic templates, each consisting of a fact, a question, and two pools of related-but-non-redundant mentions (one "high-density" pool of four mentions, one "low-density" pool of two).

Template 1: Project Aurora.

- Fact: "The secret code for Project Aurora is ⟨value⟩."
- Question: "What is the secret code for Project Aurora?"
- High-density mentions: "The team discussed Project Aurora's timeline and milestones." / "Progress reports for Project Aurora were reviewed by management." / "The Aurora initiative has been a key focus this quarter." / "Resources were reallocated to support Project Aurora's goals."
- Low-density mentions: "Various projects were discussed in the meeting." / "The quarterly review covered several ongoing initiatives."

Template 2: Agent Nightingale.

- Fact: "Agent Nightingale's extraction point is ⟨value⟩."
- Question: "What is Agent Nightingale's extraction point?"

Table 14: Complete CrystalCache default configuration (`configs/default.yaml`). Defaults are unchanged across all reported experiments.

Group	Parameter	Default	Note
Cache budget	`budget_ratio` (β)	$\{0.3, 0.5\}$	evaluated at both
	`sink_tokens` (N_{sink})	4	matches StreamingLLM
	`recent_window` (W_{recent})	128	—
Trunk construction	`max_trunk_size` ($T_{\max}$)	32	hard upper bound on $\lvert g \rvert$
	`merge_threshold` (τ_{merge})	0.30	on CAS, Eq. (18)
Co-attention edges	`prefill_chunk_size` (C)	1024	also bounds eager attn peak
	`intra_layer_k` (k_{intra})	8	top-k per query, intra-chunk
	`intra_layer_min_sim` (τ_{intra})	0.30	cosine threshold
	cross-chunk k_{cross}	4	top-k source positions
	cross-chunk τ_{cross}	0.02	raw attention threshold
	`min_association` (τ_{edge})	0.05	trunk-graph edge prune
M_i computation	`strategy`	"`entropy`" (legacy)	overridden by V-R mixture
	`von_restorff_enabled`	true	invokes Eq. (22)
	S_i top-k heads	3	Eq. (11)
	`mi_min`, `mi_max` ($M_{\min}, M_{\max}$)	$0.1, 20.0$	clip range for M_i
	mixture weights (w_{S_i}, w_{U_i})	$0.5, 0.5$	Eq. (22)
D initialisation	`sigmoid_steepness` (s)	5.0	Eq. (27)
	numerical floor ε on σ_{deg}	10^{-8}	guard against degenerate σ
Score composition	`alpha` (α)	1.0	weight on $\widetilde{M}_i$ path, Eq. (29)
	trunk M_i aggregation	top-3 mean	Eq. (23)
Branch dissolution	`min_keep_tokens` ($N_{\min}$)	3	coherence threshold per partial trunk
Decode-time (dormant)	breath interval (T_{breath})	64 steps	$>$ our `max_new_tokens`$= 50$
	`beta_0` (β_0)	0.05	Eq. (36)
	`D_c` (D_c)	2.0	Eq. (36)
	`alpha_resonance` (α_{res})	0.15	Eq. (38)
	`max_hops`	2	multi-hop BFS in resonance
	`decay_per_hop` (γ_{hop})	0.5	Eq. (37)
	`min_pulse` (τ_{pulse})	0.01	BFS termination
	`crystallize_rate` (η_{Hebb})	0.05	Eq. (39)
	`association_decay` (γ_{decay})	0.99	Eq. (40)
	activation threshold (θ_{act})	0.30	on normalised η
Decoding	`max_new_tokens` (Needle, DA)	50	per-sample generation budget
	`max_new_tokens` (LongBench)	per-subset	per LongBench-official `max_gen`
	sampling	disabled (greedy)	argmax

- High-density mentions: "Agent Nightingale reported in from the field yesterday." / "The handler confirmed Nightingale's cover remains intact." / "Updates on Nightingale's mission status were classified." / "Nightingale's next check-in is scheduled for tomorrow."
- Low-density mentions: "Field agents continued standard operations." / "Status updates were provided for all active agents."

Template 3: Formula X.

- Fact: "The activation temperature for Formula X is ⟨value⟩ degrees."
- Question: "What is the activation temperature for Formula X in degrees?"
- High-density mentions: "Formula X showed promising results in the latest trial." / "The researchers adjusted Formula X's concentration levels." / "Testing of Formula X continues in Lab 7." / "Formula X outperformed all other candidate compounds."
- Low-density mentions: "Laboratory experiments continued as scheduled." / "Multiple formulas were tested this week."

B.2 Sample Construction

A sample is constructed by the following procedure (random seed 42 for all reproducible cells):

1. Sample a template uniformly at random.
2. Sample a four-digit value uniformly from $[1000, 9999]$ and instantiate the fact.
3. Place the fact at the beginning of the context (after a brief framing sentence).
4. Generate filler text by repeating a generic-organisational paragraph until the target distance (in tokens) is reached.
5. Insert the appropriate number of related mentions (4 high-density or 2 low-density) at uniformly spaced positions throughout the filler.
6. Append the question at the end of the filler with the standard "Answer:" prompt.

B.3 Configuration Grid

- Distance (fact-to-question, in tokens): $\{4096, 8192, 16384\}$.
- Density: $\{\text{high}, \text{low}\}$.
- Samples per (distance, density) cell: 10.
- Total samples: $3 \times 2 \times 10 = 60$.

B.4 Metric

Exact-match accuracy: the predicted answer (model's first 50 generated tokens) contains the correct $\langle$value$\rangle$ as a substring.

B.5 Why This Design Probes D

The fact appears exactly once. The related mentions do *not* repeat the fact, so the rarity term U_i is unchanged whether the high-density or low-density pool is used. The mentions *do* share content with the fact's surrounding sentence ("Project Aurora," "Aurora," "Nightingale," "Formula X"), establishing co-attention edges into the fact's trunk via the cross-chunk mechanism of Section 3.2. A method that retains the fact through these structural connections (high D) should outperform a method that retains tokens only by direct attention statistics. The contrast between high-density and low-density pools is a sensitivity check: does the score follow the strength of the structural signal? Per-density CrystalCache numbers (extracted from `metrics.json` for inspection):

- Llama-3.1-8B at $\beta = 0.5$: high 0.800, low 1.000 (DA mean 0.900).
- Mistral-7B at $\beta = 0.5$: high 0.833, low 1.000 (DA mean 0.917).
- Qwen3-8B at $\beta = 0.5$: high 0.167, low 0.900 (DA mean 0.533).

The low-density pool counter-intuitively scores higher on every model. We attribute this to a weaker M_i-vs-D confound: in the high-density regime, the four related mentions themselves accumulate non-trivial M_i scores and may be retained *instead of* the fact when the budget is tight; in the low-density regime there is less competition, so the fact's trunk wins on D alone. This subtlety is exactly what makes DA an informative diagnostic: the contribution of D is most visible where competing M_i-driven candidates are scarce.

C LongBench Per-Subset Breakdown

This appendix gives the per-subset LongBench v1 scores at $\beta = 0.5$ that underlie the headline mean reported in Table 7. Numbers are exact-match for retrieval-style subsets, F1 for QA, ROUGE-L for summarisation, and accuracy for classification, per the LongBench-official metric for each subset.

Where CrystalCache loses (and where it doesn't). The pattern across all three models is consistent. CrystalCache loses most heavily on `triviaqa` (Llama: -0.10 vs. PyramidKV; Qwen3: -0.27 vs. ChunkKV) and `hotpotqa` (Llama: -0.10 vs. ChunkKV)—tasks whose answer-relevant content is spread across many spans, where token-level methods that maintain spatially uniform sampling do better. CrystalCache wins or ties on `trec` (a classification task where the candidate label list is a structurally important high-D region that CrystalCache preserves) and `qasper` (Mistral, Qwen3—scientific QA where the answer is in a specific paragraph that the trunk graph identifies).

Table 15: Llama-3.1-8B LongBench v1 per-subset scores at $\beta = 0.5$. Bold marks the best non-FullCache value per row.

Subset	Full	StrLLM	H2O	SnapKV	PyrKV	ChunkKV	**CC**
narrativeqa	0.258	0.056	0.044	0.087	**0.108**	0.068	0.051
qasper	0.232	0.030	0.080	0.069	0.054	**0.081**	0.061
multifieldqa_en	0.351	0.075	0.094	0.099	0.104	**0.122**	0.084
hotpotqa	0.083	0.067	0.069	0.084	0.109	**0.194**	0.097
2wikimqa	0.102	0.127	0.048	**0.167**	0.063	0.080	0.088
musique	0.130	0.044	0.105	0.089	**0.212**	0.051	0.043
trec	0.750	0.450	0.400	0.500	0.400	0.550	**0.600**
triviaqa	0.734	0.134	0.207	0.232	**0.312**	0.209	0.214
samsum	0.065	**0.082**	0.062	0.065	0.075	0.073	0.052
passage_count	0.100	0.000	0.050	**0.100**	0.050	**0.100**	0.000
passage_retrieval_en	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Mean	**0.255**	0.097	0.105	0.136	0.135	**0.139**	0.117

Table 16: Mistral-7B-Instruct-v0.3 LongBench v1 per-subset scores at $\beta = 0.5$. Bold marks the best value per row.

Subset	H2O	SnapKV	ChunkKV	**CC**
narrativeqa	0.060	**0.081**	0.046	0.029
qasper	0.077	0.091	**0.166**	0.119
multifieldqa_en	0.129	**0.135**	0.107	0.071
hotpotqa	0.105	**0.131**	0.117	**0.131**
2wikimqa	0.087	**0.089**	0.062	0.036
musique	0.058	**0.089**	0.082	0.035
trec	0.600	**0.650**	**0.650**	**0.650**
triviaqa	0.078	**0.174**	0.135	0.153
samsum	0.039	0.032	0.025	**0.048**
passage_count	0.100	**0.200**	0.050	0.100
passage_retrieval_en	0.000	0.000	0.000	0.000
Mean	0.121	**0.152**	0.131	0.125

The `passage_retrieval_en` zero is universal. All methods including FullCache score 0.000 on `passage_retrieval_en`. This is consistent with documented LongBench evaluation issues for this subset under instruction-tuned models that prepend chat templates: the official metric expects the model to return the exact paragraph identifier, but instruction-tuned Llama-3.1-8B / Mistral-7B-v0.3 / Qwen3-8B all wrap the answer in surrounding text. The zero is therefore a metric-format artefact, not a method failure. We retain it in the per-subset table for transparency but note that it depresses every method's mean by approximately 0.009.

D Full Prefill Pipeline Pseudocode

Algorithm 2 gives the complete prefill pipeline as it is realised in `crystalcache/integration/hf_wrapper.py`, integrating the six stages S1–S6 of Section 3. References to numbered equations are to the main text.

Implementation entry points.

- Top-level orchestration: `CrystalCacheWrapper.prefill` (`crystalcache/integration/hf_wrapper.py`, lines 110–278).
- Trunk construction: `build_trunks_from_sentences` (`crystalcache/core/trunk.py`, lines 232–307).
- Trunk graph and D: `TrunkGraph.build_from_token_edges` and `TrunkGraph.initialize_d` (`crystalcache/graph/trunk_graph.py`, lines 32–69, 153–175).
- Branch dissolution: `TrunkTable.branch_dissolve` (`crystalcache/core/trunk.py`, lines 103–209).
- Cache compression: `CrystalCacheWrapper._compact_kv_cache` (`crystalcache/integration/hf_wrapper.py`, lines 284–320).

The decode-time mechanism (Section 3.8, Equations (36)–(40)) is implemented in `decode_step` of the same wrapper but does not fire in our experiments since $T_{\text{breath}} = 64 >$ `max_new_tokens`$= 50$.

Table 17: Qwen3-8B LongBench v1 per-subset scores at $\beta = 0.5$. Bold marks the best value per row.

Subset	H2O	SnapKV	ChunkKV	**CC**
narrativeqa	0.045	0.045	**0.049**	0.043
qasper	0.037	0.024	0.028	**0.075**
multifieldqa_en	0.077	**0.112**	0.083	0.077
hotpotqa	0.059	**0.089**	0.064	0.060
2wikimqa	**0.150**	0.121	0.076	0.073
musique	0.005	0.005	0.008	**0.009**
trec	0.500	**0.550**	**0.550**	0.500
triviaqa	0.407	0.430	**0.567**	0.298
samsum	0.014	0.020	**0.043**	0.028
passage_count	0.050	0.050	0.050	0.050
passage_retrieval_en	0.000	0.000	0.000	0.000
Mean	0.122	0.131	**0.138**	0.110

Algorithm 2 Complete CrystalCache prefill pipeline.

Require: token ids $\mathbf{t} \in \mathbb{N}^n$, model $\mathcal{M}$, tokenizer $\mathcal{T}$, retention budget β
Ensure: compressed KV cache, internal trunk and graph state for decode

// Stage S1: chunked eager forward, salience, edges
$E \leftarrow \emptyset$; $S_i^{(\cdot)} \leftarrow \mathbf{0}$; past_kv $\leftarrow \emptyset$
for each chunk $[\,c, c+C\,)$ of $\mathbf{t}$ ($C = 1024$) **do**
  temporarily switch first-layer attention to eager
  out $\leftarrow \mathcal{M}(\mathbf{t}[c{:}c{+}C]$, past_kv, return_attn $=$ True$)$
  $A^{(1)} \leftarrow$ out.attentions[0] ▷ $\in \mathbb{R}^{H\times Q\times K}$
  revert attention implementation to SDPA
  for each query position q in chunk **do**
    $S_i \leftarrow \mathrm{clip}\big(\sum_{h\in \mathrm{Top3}(h^{(q)})} h_h^{(q)}, M_{\min}, M_{\max}\big)$ ▷ Eq. (11)
  end for
  $\bar{A} \leftarrow A^{(1)}$ averaged over heads; $\bar{A}_{\text{intra}} \leftarrow \bar{A}[\,\cdot, c{:}c{+}C\,]$
  for each i in chunk **do** ▷ intra-chunk row-cosine edges
    $\tilde{A}_i \leftarrow \bar{A}_{\text{intra},i,\cdot}/\|\bar{A}_{\text{intra},i,\cdot}\|_2$
  end for
  $E_{\text{intra}} \leftarrow$ top-k_{intra} of $\langle \tilde{A}_i, \tilde{A}_j\rangle > \tau_{\text{intra}}$ per i
  if $c > 0$ **then** ▷ cross-chunk raw attention edges
    $E_{\text{cross}} \leftarrow$ top-k_{cross} of $\bar{A}[\,\cdot, 0{:}c\,] > \tau_{\text{cross}}$ per query
  end if
  $E \leftarrow E \cup E_{\text{intra}} \cup E_{\text{cross}}$
  past_kv $\leftarrow$ out.past_key_values
end for

// Stage S3 (M_i finalisation, requires token-frequency)
$\{c_i\}_{i=0}^{n-1} \leftarrow$ token-frequency counter on $\mathbf{t}$
for each i **do**
  $U_i \leftarrow 1/(1+\log(1+c_i))$ ▷ Eq. (21)
  $\widetilde{S}_i \leftarrow S_i/M_{\max}$
  $M_i \leftarrow \mathrm{clip}\big(M_{\max}\cdot(\frac{1}{2}\widetilde{S}_i + \frac{1}{2}U_i),\ M_{\min}, M_{\max}\big)$ ▷ Eq. (22)
end for

// Stage S2: trunk construction
$\{s_1, \ldots, s_M\} \leftarrow$ split $\mathbf{t}$ at sentence-end tokenizer ids (`. ! ? \n`)
$g_{\text{cur}} \leftarrow s_1$; trunks $\leftarrow [\,]$
for $i = 2, \ldots, M$ **do**
  compute $\mathrm{CAS}(g_{\text{cur}}, s_i)$ via Eq. (18) on the interface window
  if CAS $> \tau_{\text{merge}}$ and $|g_{\text{cur}}| + |s_i| \le T_{\max}$ **then**
    $g_{\text{cur}} \leftarrow g_{\text{cur}} \cup s_i$
  else
    commit g_{cur} to trunks; $g_{\text{cur}} \leftarrow s_i$
  end if
end for
commit g_{cur}
for each trunk g with $|g| > T_{\max}$ **do**
  split into $\lceil |g|/T_{\max} \rceil$ pieces of equal size
end for
for each trunk g **do**
  $\overline{M_i}(g) \leftarrow$ mean of top-3 member M_i values ▷ Eq. (23)
end for

// Stage S4: trunk graph and D initialisation
aggregate E into $W(g_a, g_b)$ via Eq. (25) (with prune τ_{edge})
for each trunk g **do**
  $\deg(g) \leftarrow \sum_{g'\neq g} W(g, g')$; standardise to $z(g) = (\deg(g) - \mu)/\max(\sigma, \varepsilon)$
  $D(g) \leftarrow \sigma_{\text{logistic}}(s \cdot z(g))$ ▷ Eq. (27)
end for

// Stages S5 + S6: score composition and branch dissolution
$B \leftarrow \max(N_{\text{sink}} + W_{\text{recent}}, \lceil \beta n \rceil)$ ▷ Eq. (30)
Protected $\leftarrow$ trunks containing a token in $[0, N_{\text{sink}}) \cup [n - W_{\text{recent}}, n)$
$\mathcal{R}, \mathcal{D} \leftarrow$ Algorithm 1 on (trunks, scores, B)
for each layer ℓ: $K^{(\ell)} \leftarrow K^{(\ell)}[\mathcal{R}]$; $V^{(\ell)} \leftarrow V^{(\ell)}[\mathcal{R}]$ ▷ Eq. (35)
rebuild RoPE position-id tensor from $\mathcal{R}$
return compressed cache (past_kv), trunk table, trunk graph