International Journal of
Network Dynamics and Intelligence (IJNDI)

Scilight

Article

Attention-Derived Dependency Cut for Independent KV-Cache Reuse in Agent Memory

Yizhuo Zhao, Boxi Wu *, and Xiaofei He

Department of Engineering, Imperial College London, London SW7 2AZ, UK
* Correspondence: wuboxi@zju.edu.cn
Received: 1 September 2026; Revised: 28 September 2026; Accepted: 29 October 2026; Published: 6 November 2026

Abstract: Long-term agent memory grows continuously with interaction, making full-memory prefill increasingly costly in terms of time to first token (TTFT) and redundant computation. KV-cache reuse could avoid repeatedly prefilling the accumulated memory, but KV caching relies on exact prefix matching. When memory is updated, reorganized, or assembled from reusable units, prefix mismatches prevent direct reuse of existing KV caches and force the model to recompute a long memory prefix. Existing methods largely assume that memory blocks are already given and address the prefix-matching limitation by concatenating their KV caches and then repairing the resulting mismatch. We therefore argue that the primary question for agent-memory KV reuse is where to segment the memory. We propose AD-CUT, an Attention-derived dependency Cut method for training-free, independence-oriented segmentation. AD-Cut computes attention-derived dependency signals over sentence-level atomic units, including an incoming signal A_{in} and an outgoing signal A_{out}, and selects boundaries where both signals are low. The signals exhibit stable positive associations with intervention-derived KV and NLL sensitivity, with Spearman ρ ranging from 0.615 to 0.868 and context-cluster bootstrap intervals remaining positive, supporting their use as predictive proxies. In block-KV serving for agent QA, AD-Cut achieves a $16.17\times$ Online TTFT speedup with a QA Quality Retention of 0.7675. It also improves task quality over naive sentence-level reuse in a controlled agent-memory evaluation. These results show that AD-Cut substantially reduces online computation while mitigating the quality loss caused by KV-cache reuse.

Keywords: agent memory; KV-cache reuse; attention dependency; memory segmentation; large language model serving

1. Introduction

Long-term agents must retain user preferences, prior tasks, environment states, workflow knowledge, and interaction trajectories. As interactions accumulate, agent memory grows from a short context into a long-term memory stream [1,2]. This growth increases the full-memory attention and prefill costs of each request [3–6], as illustrated in Figure 1a.

The central efficiency challenge in long-term agent serving is to avoid repeated computation over a growing memory context. KV caching is commonly used to address this problem. It stores the key-value states produced during prefill and reuses them during subsequent generation, thereby reducing redundant computation [3,7–10]. The most direct strategy is to cache and reuse the complete memory prefix.

This strategy is effective when the memory prefix remains unchanged. However, long-term agent memory may be updated or reorganized as interactions accumulate. Because KV-cache reuse relies on prefix matching, a change to an earlier part of the context causes the KV states of subsequent tokens to no longer correspond to the updated context, requiring them to be recomputed through a new prefill [11,12]. Each request may therefore still fall back to recomputing the complete relevant memory, causing TTFT, prefill FLOPs, and GPU memory-bandwidth costs to increase with memory length [3,4].

To overcome the prefix-matching limitation of complete-memory KV caching, existing block-KV reuse methods assume that the memory has already been organized into predefined blocks. These blocks are prefilled and cached separately, and their KV caches are concatenated and reused at query time instead of prefilling the complete memory again [3,11].

However, blockwise KV reuse introduces a block-independence mismatch. Independently prefilling two adjacent blocks and then concatenating their KV caches is generally not equivalent to jointly prefilling them [3,4,12]. As illustrated in Figure 1b, independent prefill removes the cross-block attention that is available during full-context prefill. It may also alter positional contextualization, textual continuity, and attention-sink distributions, which can reduce generation quality [3,4,7]. This issue is particularly relevant to agent memory, where one memory unit may

Copyright: © 2026 by the authors. This is an open access article under the terms and conditions of the Creative Commons Attribution (CC BY) license(https://creativecommons.org/licenses/by/4.0/).
Publisher's Note: Scilight stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

depend on an earlier state and later units may inherit, revise, or overwrite that state [1,2]. Applying block-KV reuse to agent memory therefore requires low-dependency boundaries rather than arbitrary predefined blocks.

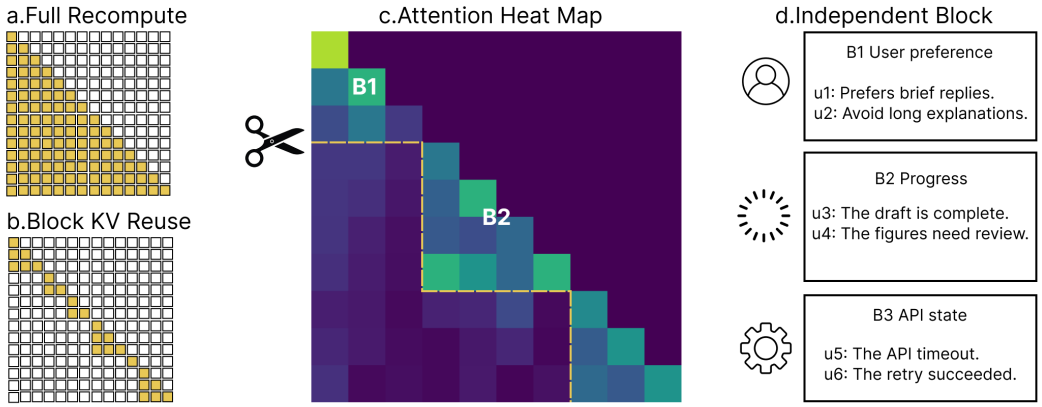


Figure 1. Motivation for dependency-aware agent-memory segmentation. (**a**) Full Recompute performs causal prefill over the complete memory context, so its computation grows with context length; (**b**) Independent Block KV Reuse preserves within-block attention but cannot retain the cross-block attention available in full-context prefill; (**c**) The attention heat map visualizes the dependency structure among atomic units. The yellow dashed lines mark candidate boundaries in regions with weaker cross-block attention; (**d**) AD-Cut uses these boundaries to form fine-grained independent blocks rather than following predefined memory categories.

Existing KV-reuse methods primarily focus on cache repair or selective recomputation after blocks have been defined. Block-Attention addresses positional alignment in block-KV reuse [11]. KVLink introduces linking tokens to restore part of the cross-block attention [4]. KV Packet uses soft-token adapters, while EPIC recomputes block-initial tokens to mitigate prefix mismatch and attention sinks [7,12]. CacheBlend fuses cached chunks by recomputing a small set of important tokens [3]. Recent segmentation methods such as MoC and SemanticSeg address boundary selection, but they focus on text-level chunking quality and semantic self-containment rather than model-internal independence under separate prefill [13,14].

To address this gap, we propose AD-Cut, a training-free method that identifies candidate low-dependency boundaries from the attention structure obtained during full-context prefill, as illustrated in Figure 1c. Its central idea is that a boundary is preferable when the unit after it receives little attention from the preceding context and the unit before it contributes little attention to the future context. AD-Cut uses the incoming signal A_{in} and the outgoing signal A_{out} as proxies for these dependencies over sentence-level atomic units. It applies a low-incoming and low-outgoing rectangle rule to construct fine-grained memory blocks suitable for independent prefill, caching, and reuse, as shown in Figure 1d.

Our contributions are threefold. First, we shift the central question in agent-memory KV reuse from how to compose cached states to where the memory should be segmented. Second, we introduce attention-derived dependency signals and validate their associations with intervention-derived KV and NLL independence errors in Q1. Third, we develop AD-Cut and evaluate its boundary quality, independence measures, Online TTFT, and agent-memory QA quality in Q2 and Q3.

2. Methods

2.1. Problem Formulation

Given an agent memory M, our goal is to partition it into KV blocks that can be independently prefilled, cached, and reused at query time without additional training.

We represent M as an ordered sequence U of sentence-level atomic units:

$$U = (u_1, u_2, \ldots, u_N), \tag{1}$$

where u_i is the i-th sentence-level atomic unit, and N is the number of atomic units in M. Candidate boundaries are restricted to adjacent units. We denote the boundary between u_t and u_{t+1} by $B_t = u_t \mid u_{t+1}$, where $t \in \{1, \ldots, N-1\}$. The offline stage constructs a KV cache for each resulting block. At query time, the relevant block KV caches are loaded and combined to reduce repeated prefilling of the full memory prefix.

2.2. Attention-Derived Dependency Signals

We view the error introduced by independent block prefilling as related to removing cross-boundary dependencies [3,4,12]. To construct proxies for the dependency around candidate boundary $B_t = u_t \mid u_{t+1}$, AD-Cut defines two attention-derived signals. $A_{\text{in}}(t)$ summarizes attention from u_{t+1} to the prefix context $U_{\leq t}$. $A_{\text{out}}(t)$ summarizes attention from future units to u_t . A boundary is treated as more suitable for independent prefilling and caching when both signals are low. The construction of the incoming and outgoing dependency signals is illustrated in Figure 2a.

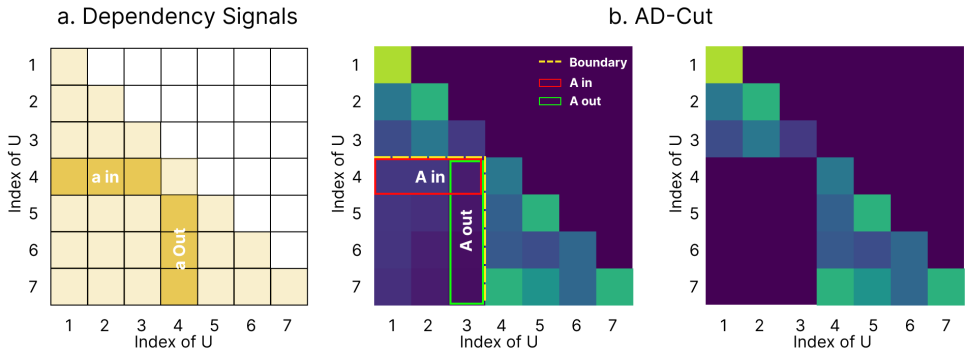


Figure 2. Overview of AD-Cut dependency signals and boundary selection. (**a**) For sentence-level atomic unit u_4, $a_{\text{in}}(4)$ measures the attention that u_4 assigns to the historical units u_1, u_2, and u_3. The signal $a_{\text{out}}(4)$ measures the attention that the future units u_5, u_6, and u_7 assign to u_4. (**b**) For candidate boundary $B_3 = u_3 \mid u_4$, $A_{\text{in}}(3)$ is obtained by normalizing $a_{\text{in}}(4)$ from the right-side unit. The signal $A_{\text{out}}(3)$ is obtained by transforming and normalizing $a_{\text{out}}(3)$ from the left-side unit. AD-Cut selects the boundary only when both signals satisfy their thresholds and the structural guard accepts the cut. The rightmost panel shows the within-block attention retained after independent block prefill. Cross-block attention is unavailable.

Unit-level attention matrix. To compute these signals, AD-Cut aggregates the token-level attention obtained from full-context prefill into a unit-level attention matrix W. Let T_i denote the token set corresponding to u_i. Let $\alpha_{\ell,h}(q,k)$ denote the attention weight assigned by query token q to source token k at layer ℓ and attention head h [15]. We define

$$W[j,i] = \frac{1}{LH|T_j|} \sum_{\ell=1}^{L} \sum_{h=1}^{H} \sum_{q \in T_j} \sum_{k \in T_i^\star} \alpha_{\ell,h}(q,k), \qquad i < j, \tag{2}$$

where L is the number of Transformer layers, H is the number of attention heads per layer, and $T_i^\star$ is the source-token set used for attention aggregation. Thus, $W[j,i]$ represents the average attention mass that u_j assigns to u_i.

AD-Cut uses a sink-controlled attention matrix to construct the boundary features. If u_i is the first atomic unit in its source memory block, $T_i^\star$ is obtained by removing the first source token of u_i from T_i while retaining all remaining tokens. Otherwise, $T_i^\star = T_i$. This operation only reduces the influence of block-first attention sinks on the boundary signals [7,16].

2.2.1. Signal Processing

AD-Cut processes the incoming and outgoing boundary features separately within each memory sample. We denote the sample-level min–max normalization that maps a feature to $[0,1]$ by $\text{Norm}(\cdot)$. The two signals are not jointly scaled. Their transformations are defined below.

2.2.2. Incoming Dependency Signal

For atomic unit u_j, the raw incoming signal is defined as

$$a_{\text{in}}(j) = \sum_{i=1}^{j-1} W[j,i], \qquad j \in \{2,\ldots,N\}. \tag{3}$$

For candidate boundary $B_t = u_t \mid u_{t+1}$, the normalized incoming boundary signal is

$$A_{\text{in}}(t) = \text{Norm}\left(a_{\text{in}}(t+1)\right), \qquad t \in \{1, \ldots, N-1\}. \tag{4}$$

Equation (3) measures the attention that u_j assigns to its historical context $U_{<j}$ and provides the raw unit-level incoming signal used in Q1. Each $a_{\text{in}}(j)$ aggregates the nonnegative attention mass that query tokens in u_j assign to historical atomic units. Since attention weights are normalized over all source positions, $a_{\text{in}}(j)$ lies in $[0, 1)$. Thus, $\text{Norm}(\cdot)$ in Equation (4) applies a simple sample-level min–max normalization. For Q2, AD-Cut normalizes the raw signal of the first atomic unit after the boundary to obtain $A_{\text{in}}(t)$.

2.2.3. Outgoing Dependency Signal

For candidate boundary $B_t = u_t \mid u_{t+1}$, the raw outgoing feature of the left atomic unit is defined as

$$a_{\text{out}}(t) = \frac{1}{N-t} \sum_{j=t+1}^{N} W[j, t]. \tag{5}$$

This feature measures the average attention assigned to u_t by the future units $U_{\geq t+1}$. Averaging over future units controls for the different numbers of future units available at different positions.

The distribution of $a_{\text{out}}(t)$ is typically skewed, and a small number of large values may compress the min–max range of the remaining boundaries. AD-Cut therefore applies positive-median scaling followed by a logarithmic transformation and sample-level normalization:

$$\begin{aligned} m^{+}_{\text{out}} &= \text{median}\left(\{a_{\text{out}}(s) \,|\, a_{\text{out}}(s) > 0\}\right), \\ z_{\text{out}}(t) &= \log\left(1 + \frac{a_{\text{out}}(t)}{m^{+}_{\text{out}}}\right), \\ A_{\text{out}}(t) &= \text{Norm}\left(z_{\text{out}}(t)\right). \end{aligned} \tag{6}$$

If no positive value exists, we set $m^{+}_{\text{out}} = 1$. Positive-median scaling calibrates the outgoing attention scale across samples, while the logarithmic transformation reduces the effect of long-tailed values. The final normalization produces the boundary signal $A_{\text{out}}(t)$ used for boundary selection.

2.2.4. AD-Cut Boundary Selection

Figure 2b provides an overview of the AD-Cut boundary-selection process. For each candidate boundary B_t, AD-Cut applies a low-incoming and low-outgoing rectangle rule in the dependency space defined by $A_{\text{in}}(t)$ and $A_{\text{out}}(t)$:

$$\text{cut}(B_t) = \mathbb{I}\left[A_{\text{in}}(t) \leq \tau_{\text{in}} \wedge A_{\text{out}}(t) \leq \tau_{\text{out}} \wedge g_{\text{struct}}(u_t, u_{t+1}) = 1\right]. \tag{7}$$

Here, τ_{in} and τ_{out} are the decision thresholds for the normalized incoming and outgoing signals, and g_{struct} denotes the structural guard. A boundary is selected only when dependency is low in both directions and the structural guard accepts the cut. Independent prefill then retains attention within each resulting block while removing attention across the selected boundary, as shown in the rightmost heat map of Figure 2b.

The structural guard is an explicit text-pattern filter that rejects boundaries involving standalone list markers, marker-content separation, short fragments, and other structurally invalid splits. Its full rule set is provided in Appendix A.

A 1000-sample development-only sweep selected $\tau_{\text{in}} = 0.5$ and $\tau_{\text{out}} = 0.4$ before evaluation on the 2000-sample outer test.

2.3. Independence Measures

AD-Cut seeks boundaries suitable for independent block prefill. We evaluate segmentation independence using local representation and behavior errors together with segmentation-level cross-block attention [3,4].

2.3.1. Local Independence Errors

We measure local dependency in two directions. Focal-unit errors measure the dependency of u_j on its historical context. Future-unit errors measure the influence of u_j on subsequent units.

Let $\text{KV}(u)$ denote the Key–Value states of atomic unit u. Let $\text{NLL}(u)$ denote the mean token-level negative log-likelihood over the valid tokens covered by u. The subscript full denotes full-context prefill, while iso denotes isolated prefill. The subscript $\setminus j$ denotes a masked-source intervention in which the token IDs covered by u_j are

replaced with a fixed mask-surrogate token. The sequence length, all remaining tokens, and their position indices are preserved. These subscripts apply to both KV states and NLL values.

The local independence errors are defined as

$$\begin{aligned}
\mathcal{E}^{\mathrm{KV}}_{\mathrm{focal}}(j) &= d_{\mathrm{KV}}\left(\mathrm{KV}_{\mathrm{full}}(u_j), \mathrm{Align}\left[\mathrm{KV}_{\mathrm{iso}}(u_j)\right]\right), \\
\mathcal{E}^{\mathrm{NLL}}_{\mathrm{focal}}(j) &= \mathrm{NLL}_{\mathrm{iso}}(u_j) - \mathrm{NLL}_{\mathrm{full}}(u_j), \\
\mathcal{E}^{\mathrm{KV}}_{\mathrm{future}}(j) &= \frac{1}{N-j}\sum_{i=j+1}^{N} d_{\mathrm{KV}}\left(\mathrm{KV}_{\mathrm{full}}(u_i), \mathrm{KV}_{\setminus j}(u_i)\right), \\
\mathcal{E}^{\mathrm{NLL}}_{\mathrm{future}}(j) &= \frac{1}{N-j}\sum_{i=j+1}^{N} \left[\mathrm{NLL}_{\setminus j}(u_i) - \mathrm{NLL}_{\mathrm{full}}(u_i)\right].
\end{aligned} \tag{8}$$

The focal-unit errors quantify changes in u_j after removing its prefix context. The future-unit errors quantify changes in subsequent units after masking u_j. KV and NLL errors provide complementary representation-side and behavior-side evidence. Lower values indicate weaker local dependency.

The distance d_{KV} computes the relative Frobenius distance between position-aligned key and value states at each Transformer layer, then averages the distances across keys, values, and layers. The operator Align maps isolated KV states from local RoPE positions to their full-context positions so that positional differences are not counted as contextual dependency [4,17]. Future-unit errors are averaged over the $N-j$ subsequent units to control for variation in the number of available future units.

2.3.2. Robustness Analysis

The raw signal $a^{\mathrm{nf}}_{\mathrm{in}}(j)$ excludes the contribution of the first atomic unit in each source session. This variant tests whether the incoming relationship is mainly driven by attention-sink effects [7,16].

We also separate outgoing dependency into same-session and cross-session components. Let $b(i)$ denote the source session containing u_i. A future unit u_i belongs to the same-session subset when $i > j$ and $b(i) = b(j)$. It belongs to the cross-session subset when $i > j$ and $b(i) \neq b(j)$. We compute $a^{\mathrm{same}}_{\mathrm{out}}(j)$ and $a^{\mathrm{cross}}_{\mathrm{out}}(j)$ over these subsets and compare them with the corresponding $\mathcal{E}^{\mathrm{KV/NLL}}_{\mathrm{future,same/cross}}(j)$ errors.

2.3.3. Segmentation-Level Independence Measures

Local errors measure the contextual sensitivity of individual atomic units. A complete segmentation also requires a measure of the total dependency removed across blocks. Let Π denote a segmentation, and let $c_\Pi(i)$ denote the block containing u_i. The Cross-Block Attention Score is defined as

$$\mathrm{CS}(\Pi) = \sum_{i>j} W[i,j]\,\mathbb{I}\left[c_\Pi(i) \neq c_\Pi(j)\right]. \tag{9}$$

Equation (9) aggregates the cross-block attention paths removed by independent block prefill. A lower CS indicates that the segmentation removes less cross-block dependency.

Raw CS is sensitive to the number of cuts because additional cuts create more cross-block attention paths. We therefore define CS Adjusted to control for segmentation granularity. A higher CS Adjusted indicates that the segmentation removes less cross-block dependency at the same cut count. Let K denote the number of cuts. Let Π^K_{rand} denote a random segmentation with K cuts, and let Π^K_{oracle} denote the segmentation that minimizes CS under the same cut count. We define

$$\mathrm{CSAdj}(\Pi) = \frac{\mathbb{E}\left[\mathrm{CS}\left(\Pi^K_{\mathrm{rand}}\right)\right] - \mathrm{CS}(\Pi)}{\mathbb{E}\left[\mathrm{CS}\left(\Pi^K_{\mathrm{rand}}\right)\right] - \mathrm{CS}\left(\Pi^K_{\mathrm{oracle}}\right)}. \tag{10}$$

3. Experiments

3.1. Research Questions

The experiments address the following research questions:

- Q1: Dependency Signal Validation. Do the raw incoming signal a_{in} and outgoing signal a_{out} reflect an atomic unit's dependency on its preceding context and its influence on future context, respectively?
- Q2: Boundary Selection Quality. Can AD-CUT, using A_{in} and A_{out}, identify independence-oriented boundaries with weak cross-boundary dependency while preserving valid textual structure?

- Q3: Block-KV Serving Efficiency and Task Quality. When memory blocks produced by AD-CUT are independently prefetched and reused as KV caches, can they reduce online TTFT while preserving RAG QA quality and controlled agent-memory task quality?

Together, the three research questions examine the signal-level motivation, boundary-selection behavior, and serving-level utility of AD-CUT.

3.2. Q1: Dependency Signal Validation

Q1 evaluates whether the raw attention features used by AD-Cut are associated with intervention-derived local contextual sensitivity. For each sentence-level atomic unit u_j, the incoming analysis relates $a_{\text{in}}(j)$ to $\mathcal{E}^{\text{KV}}_{\text{focal}}(j)$ and $\mathcal{E}^{\text{NLL}}_{\text{focal}}(j)$. The outgoing analysis relates $a_{\text{out}}(j)$ to $\mathcal{E}^{\text{KV}}_{\text{future}}(j)$ and $\mathcal{E}^{\text{NLL}}_{\text{future}}(j)$. Larger attention signals are expected to correspond to larger independence errors. Each sentence-level atomic unit forms one observation.

3.2.1. Data and Model

Q1 uses Tulu3-SFT [18,19] for memory-only prefill and draws data from LongMemEval agent memory [1]. We construct 100 memory contexts with 11,285 KV observations. Full construction and exclusion details are provided in Appendix B.

3.2.2. Experimental Procedure

For each memory context, we run full-context prefill to extract the unit-level attention matrix, KV states. Incoming validation independently prefills each focal unit and aligns its isolated KV states to the corresponding full-context positions. Outgoing validation replaces the focal-unit token IDs with a fixed mask-surrogate token while preserving the remaining tokens and position indices, then repeats full-context prefill.

3.2.3. Comparison and Statistical Analysis

Incoming dependency is evaluated by comparing $a_{\text{in}}(j)$ with $\mathcal{E}^{\text{KV}}_{\text{focal}}(j)$ and $\mathcal{E}^{\text{NLL}}_{\text{focal}}(j)$. The no-first variant $a^{\text{nf}}_{\text{in}}(j)$ is compared with the same targets to test whether the relationship is mainly driven by source-session initial units.

Under the strict masked-source intervention, outgoing dependency is evaluated by comparing $a_{\text{out}}(j)$ with $\mathcal{E}^{\text{KV}}_{\text{future}}(j)$ and $\mathcal{E}^{\text{NLL}}_{\text{future}}(j)$. The same-session and cross-session variants are compared with their corresponding future-unit errors to examine whether the relationship holds within and across source sessions.

Association strength is measured using Spearman's ρ. Partial Spearman correlations control for position and future-unit count. We estimate 95% confidence intervals using 5000 context-cluster bootstrap replicates: each replicate samples complete memory contexts with replacement, retains every valid atomic-unit observation in each selected context, and recomputes the pooled Spearman correlation. The incoming and outgoing analyses contain 96 and 100 independent contexts, respectively.

3.3. Q2: Boundary Selection Quality

Q2 evaluates whether AD-Cut can use $A_{\text{in}}(t)$ and $A_{\text{out}}(t)$ to identify boundaries that produce blocks with greater independence. We evaluate the method along two complementary dimensions. Annotation Adjusted measures textual structure and contextual completeness using LLM boundary annotations. CS Adjusted measures the cross-block dependency removed by a segmentation.

3.3.1. Data and Model

Q2 uses Tulu3-SFT [18] for full-context prefill and extracts the unit-level attention matrix required by AD-Cut. We construct 3000 boundary-selection samples from LongMemEval agent memory [1]. A reproducible, construction-stratified, memory-unit-graph-aware split assigns 1000 samples to development and 2000 samples to an outer test. The development/test split, fold construction, random seed, and shared-memory-unit overlap audit are documented in Appendix C. Each sample contains 3 to 9 sessions and 60 to 70 normalized sentence-level atomic units, with a maximum input length of 3500 tokens. GPT-5.5 [20] labels each boundary as `must`, `good`, `acceptable`, `unacceptable`, or `bad`.

3.3.2. Experimental Procedure

We run full-context prefill for each sample and compute $A_{\text{in}}(t)$ and $A_{\text{out}}(t)$ for every candidate boundary. AD-Cut performs a complete 10×10 sweep only on the 1000 development samples. The development-tuned rule requires defined Annotation Adjusted, Mean Segments no greater than 20, and Bad Rate no greater than 0.20, then

ranks feasible settings by Annotation Adjusted, CS Adjusted, Must Recall, Mean Segments, τ_{in}, and τ_{out}. It selects $(0.5, 0.4)$; all five 200-sample development folds independently select the same pair. We freeze this configuration and evaluate it once on the 2000 outer-test samples.

3.3.3. Comparison and Evaluation

The five annotation labels define a weighted boundary score. Annotation Adjusted controls for cut count by comparing each segmentation with random and annotation-oracle segmentations under the same cut count. Higher values indicate better preservation of textual structure and contextual completeness. Must Recall measures the proportion of required boundaries that are selected. Bad Rate measures the proportion of selected boundaries labeled as `bad`. CS Adjusted measures model-side independence. Mean Segments reports segmentation granularity.

We compare AD-Cut with Fixed Length, Recursive Structure [21], TextTiling [22], the official LDA TopicTiling implementation [23], LlamaIndex Semantic Splitter with BGE-M3 [24,25], and Chonkie SemanticChunker with BGE-M3 [25,26]. Implementation details and parameters are provided in Appendix D. The annotation prompt and scoring rules are provided in Appendix E.

3.4. Q3: Block-KV Reuse Evaluation

Q3 evaluates the serving efficiency and task quality of AD-Cut blocks under KV-cache reuse. We compare three serving strategies: Full Recompute, Full Reuse Sentence-Concat, and AD-Cut Block Reuse. Task A evaluates RAG QA serving, while Task B evaluates controlled agent-memory tasks.

3.4.1. Task A: RAG QA Serving

Task A examines whether AD-Cut reduces Online TTFT relative to Full Recompute while preserving higher QA quality than naive sentence-level reuse.

Data and Model

Task A uses Tulu3-Block-FT [11,18] for block-KV inference [27]. We evaluate on four RAG QA benchmarks: 2WikiMultiHopQA [28], HotpotQA [29], Natural Questions [30], and TriviaQA [31]. We randomly select 500 samples from each dataset, yielding 2000 samples in total.

Experimental Procedure

All three serving strategies use the same inputs. Full Recompute prefills the complete documents and query for every request. Full Reuse Sentence-Concat is a naive baseline that treats each sentence as an independent block. AD-Cut Block Reuse uses the blocks selected by AD-Cut. For both reuse strategies, attention extraction, segmentation, block prefill, cache construction, and cache concatenation are completed offline. Online inference starts from an assembled KV cache, appends the same query suffix used by Full Recompute, and generates the response.

Comparison and Evaluation

Task A compares online efficiency, answer quality, and offline cost. Mean Segments reports block granularity. Offline ms measures block prefill, cache construction, and cache finalization. Attention extraction and segmentation are also performed offline but are not included in Offline ms. Cache Size reports the total KV footprint for the same context.

Online TTFT measures query-time latency. For Full Recompute, timing starts with the complete documents and query and ends when the first token is generated. For the reuse strategies, timing starts when the assembled KV cache is available and includes query processing and first-token generation. TTFT Speedup is measured relative to the mean Online TTFT of Full Recompute.

GPT-5.5 evaluates the model responses against the reference answers. We report Judge Correct for each method. For method m, Quality Retention is $|I_m|^{-1} \sum_{i \in I_m} s_{m,i}/s_{\text{FR},i}$, where $I_m = \{i : s_{\text{FR},i} > 0 \text{ and } s_{m,i} \text{ is available}\}$ and each Judge Score is in $\{0, 0.5, 1\}$. Thus zero-score Full Recompute cases are excluded as undefined ratios, and Quality Retention is not the ratio of aggregate means. Both reuse methods have 1557 eligible samples and no missing scores. We additionally report normalized answer-alias containment as a deterministic lexical check..

For direct whole-block validation, Dense Full Prefill provides the reference KV states and reference-answer NLL. We compare independently prefetched-and-concatenated Sentence-Concat and AD-Cut caches using all-token and relative KV errors computed after excluding the entire first block , together with signed and absolute answer-NLL differences under teacher forcing of the same canonical answer. Uncertainty is estimated from paired question-level bootstrap resampling, with failure and token-alignment audits.

3.4.2. Task B: Controlled Agent-Memory Ability

Task B compares the same three serving strategies in a controlled agent-memory setting. The task places the relevant fact directly in memory and measures answer quality under identical frozen constructions containing different block granularities and trajectory-derived distractors [2].

Data and Model

Task B uses Tulu3-Block-FT. The questions, category labels, and scoring functions are derived from LongMemEval-V2. The five categories are Static Recall, Dynamic Tracking, Workflow Knowledge, Environment Gotchas, and Premise Awareness. We use a controlled relevant-fact trajectory-distractor context. Each context contains one task-relevant fact block derived from the gold answer and several trajectory-derived distractor blocks. The blocks use neutral `Observation block` headings.

From a pool of 451 source questions, 433 represented parent questions yield 1000 category-balanced question–context variants (200 per category). Every variant contains one task-relevant fact block and exactly eight trajectory-derived distractor blocks.

Comparison and Evaluation

Task B computes a sample-level score using the corresponding `eval_function`. We report the mean score for each of the five categories. The overall score is the macro average of the five category means. Uncertainty is estimated with 5000 category-stratified parent-question cluster bootstrap replicates. As a sensitivity analysis, parent questions receive equal weight within each category so that unequal numbers of augmented variants do not determine the aggregate ordering.

4. Results

4.1. Q1: Attention-Derived Signals and Local Independence Errors

Table 1 summarizes the formal Q1 results. The incoming signal $a_{\text{in}}(j)$ shows the strongest association with the focal KV error, with Spearman $\rho = 0.8680$. The partial correlation remains high at 0.8652 after controlling for position. The association with the focal NLL error is also strong, with Spearman $\rho = 0.7785$ and partial $\rho = 0.7746$. These results indicate that incoming attention is a useful proxy for the KV representation change under isolated prefill and the next-token predictability gained from full-context prefill.

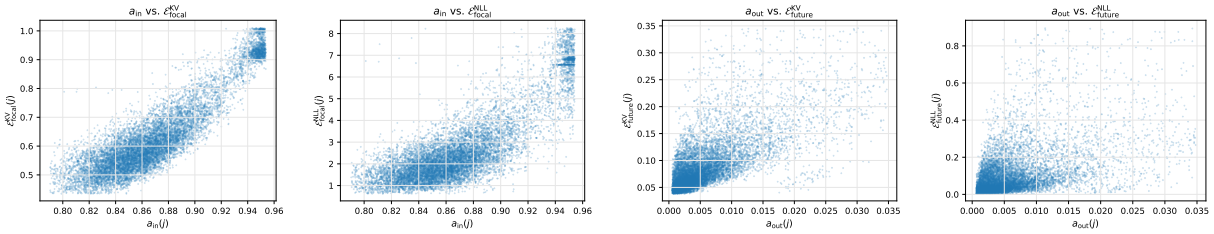
Table 1. Q1 validation of incoming and outgoing dependency signals against intervention-derived KV and NLL sensitivity. Intervals use 5000 memory-context-cluster bootstrap replicates; Obs. and Ctx. distinguish unit-level observations from independent memory contexts.

Signal	Target	Spearman ρ	Partial ρ	95% Cluster CI	Obs.	Ctx.
$a_{\text{in}}(j)$	$\mathcal{E}^{\text{KV}}_{\text{focal}}(j)$	0.8680	0.8652	[0.8484, 0.8875]	11,285	96
$a_{\text{in}}(j)$	$\mathcal{E}^{\text{NLL}}_{\text{focal}}(j)$	0.7785	0.7746	[0.7512, 0.8064]	11,278	96
$a^{\text{nf}}_{\text{in}}(j)$	$\mathcal{E}^{\text{KV}}_{\text{focal}}(j)$	0.6744	0.6758	[0.6438, 0.7045]	11,285	96
$a^{\text{nf}}_{\text{in}}(j)$	$\mathcal{E}^{\text{NLL}}_{\text{focal}}(j)$	0.6272	0.6299	[0.5963, 0.6572]	11,278	96
$a_{\text{out}}(j)$	$\mathcal{E}^{\text{KV}}_{\text{future}}(j)$	0.7510	0.6333	[0.6887, 0.7923]	11,636	100
$a_{\text{out}}(j)$	$\mathcal{E}^{\text{NLL}}_{\text{future}}(j)$	0.6147	0.4564	[0.5845, 0.6398]	11,636	100
$a^{\text{same}}_{\text{out}}(j)$	$\mathcal{E}^{\text{KV}}_{\text{future,same}}(j)$	0.8022	0.5317	[0.7595, 0.8345]	11,448	100
$a^{\text{same}}_{\text{out}}(j)$	$\mathcal{E}^{\text{NLL}}_{\text{future,same}}(j)$	0.6958	0.3920	[0.6745, 0.7156]	11,448	100
$a^{\text{cross}}_{\text{out}}(j)$	$\mathcal{E}^{\text{KV}}_{\text{future,cross}}(j)$	0.6526	0.6996	[0.5730, 0.7278]	6689	87
$a^{\text{cross}}_{\text{out}}(j)$	$\mathcal{E}^{\text{NLL}}_{\text{future,cross}}(j)$	0.2555	0.2511	[0.1895, 0.3247]	6689	87

We further evaluate $a^{\text{nf}}_{\text{in}}(j)$ to test whether the incoming relationship is mainly driven by the first atomic unit in each source session. After excluding these contributions, the Spearman correlations remain positive at 0.6744 for the KV target and 0.6272 for the NLL target.

The outgoing signal provides complementary evidence for downstream influence. Its Spearman correlation is 0.7510 with the future KV error and 0.6147 with the future NLL error. Higher outgoing attention is therefore associated with larger future KV and NLL changes under the masked-source intervention. The same-session correlations are 0.8022 for KV and 0.6958 for NLL. The cross-session correlations remain positive at 0.6526 for KV and 0.2555 for NLL, although the relationships are stronger within the same session. The memory-context-cluster confidence intervals in Table 1 remain positive for all ten reported associations.

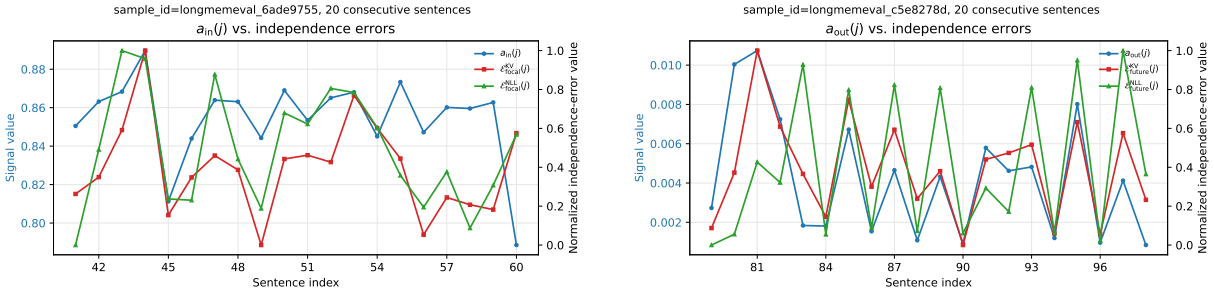
Figure 3 visualizes the primary Q1 relationships. The incoming signal shows clear positive associations with both focal KV and NLL errors. The outgoing signal also shows positive associations with the future errors. These relationships are clearer below approximately 0.015, while the observations become more dispersed at larger values.



(a) a_{in} vs. $\mathcal{E}^{\text{KV}}_{\text{focal}}$ **(b)** a_{in} vs. $\mathcal{E}^{\text{NLL}}_{\text{focal}}$ **(c)** a_{out} vs. $\mathcal{E}^{\text{KV}}_{\text{future}}$ **(d)** a_{out} vs. $\mathcal{E}^{\text{NLL}}_{\text{future}}$

Figure 3. Validation of the Q1 dependency signals. The scatter plots display the central 95% range of each axis. All statistics reported in Table 1 are computed from the complete formal observations and interventions.

Figure 4 provides a local view of the relationship between the signals and independence errors. The incoming signal follows the broad trends in the focal KV and NLL errors. The outgoing signal is more sensitive to local peaks and troughs in the future errors, which helps identify positions where downstream dependency becomes stronger or weaker.



(a) $a_{\text{in}}(j)$ and focal independence errors **(b)** $a_{\text{out}}(j)$ and future independence errors

Figure 4. Local signal and independence-error profiles across 20 consecutive sentence-level atomic units. The left axis shows the raw attention-derived signal, while the right axis shows the window-normalized KV and NLL independence errors.

4.2. Q2: Boundary Selection Quality

4.2.1. Boundary Benchmark

As shown in Table 2, Chonkie has the highest Annotation Adjusted score (0.2364), whereas AD-Cut has the highest CS Adjusted score (0.2113) and a lower Bad Rate (0.1872 versus 0.2406). Relative to the TextTiling, AD-Cut's Annotation Adjusted difference is 0.00258 with a 5000-replicate paired-bootstrap 95% CI of $[-0.01077, 0.01615]$ over 1999 jointly defined samples. Thus, their annotation quality is comparable, while AD-Cut has higher Must Recall, lower Bad Rate, and higher CS Adjusted. Overall, the held-out results support a favorable trade-off.

4.2.2. Structural Guard Clarification

The structural guard filters punctuation-induced marker-only units and short fragments from the candidate boundaries identified by the attention-based rule. Its complete rule set and first-match pseudocode are provided in Appendix A.

Table 2. Held-out Q2 comparison on the 2000-sample outer test. Thresholds and baseline configurations were frozen before test evaluation.

Method	Annotation Adjusted	Must Recall	Bad Rate	CS Adjusted	Mean Segments
AD-Cut	0.1405	0.4089	**0.1872**	**0.2113**	18.79
Fixed Length	0.0378	0.0800	0.2814	−0.1416	5.01
Recursive Structure	0.0119	0.1020	0.3538	−0.1176	5.17
TextTiling	0.1380	0.2232	0.4009	0.1203	5.29
Official LDA TopicTiling	0.0788	0.3524	0.4543	−0.0123	10.29
LlamaIndex Semantic Splitter	0.1130	0.1341	0.2871	−0.0443	5.00
Chonkie SemanticChunker	**0.2364**	**0.4975**	0.2406	−0.1151	13.70

Note: Bold values indicate the best value for each metric.

4.2.3. Human Validation of Boundary Labels

One human annotator blindly relabeled 100 frozen boundaries, stratified as 20 per GPT-5.5 label. Five-class exact agreement was $67/100 = 0.6700$ (95% CI $[0.5800, 0.7500]$), and quadratic-weighted Cohen's κ was 0.8039 (95% CI $[0.7273, 0.8707]$). After mapping labels to positive, neutral, and negative cut directions, exact agreement was 0.7400; among the 80 GPT-nonneutral items, strict positive/negative agreement was $63/80 = 0.7875$.

4.3. Q3: Block-KV Reuse Evaluation

4.3.1. Task A: RAG QA Serving

Table 3 summarizes the aggregate serving latency and QA quality results. Full Recompute provides the answer-quality baseline with a Judge Correct score of 0.7440, but requires 448.8 ms of Online TTFT. Full Reuse Sentence-Concat and AD-Cut Block Reuse reduce Online TTFT to 27.4 ms and 27.8 ms, corresponding to speedups of $16.35\times$ and $16.17\times$.

Table 3. Q3 Task A comparison of online serving latency, offline processing cost, cache footprint, and judge-based QA quality for full recomputation and block-KV reuse strategies.

Method	Mean Seg.	Offline ms	Cache MB	TTFT ms	TTFT Speedup	Judge Correct	Quality Ret.
Full Recompute	**1.00**	**446.1**	**1218.7**	448.8	1.00×	**0.7440**	**1.0000**
Full Reuse Sentence-Concat	74.29	1352.8	**1218.7**	**27.4**	**16.35**×	0.5690	0.7187
AD-Cut Block Reuse	42.37	856.1	**1218.7**	27.8	16.17×	0.6035	0.7675

Note: Bold values indicate the best value for each metric.

Full Reuse Sentence-Concat is slightly faster but obtains a Judge Correct score of 0.5690 and a Quality Retention of 0.7187. AD-Cut improves these values to 0.6035 and 0.7675 while maintaining similar Online TTFT.

All three methods have the same Cache Size of 1218.7 MB because they process the same cache tokens with the same model and KV layout. AD-Cut reorganizes the cached states into blocks without removing tokens or compressing the KV states.

Table 4 shows that AD-Cut retains more QA quality than Full Reuse Sentence-Concat on all four datasets. The average context lengths are approximately 1339 cache tokens for 2Wiki and 1682 for HotpotQA, compared with 3341 for NQ and 3388 for TriviaQA. AD-Cut achieves speedups between $7.5\times$ and $9.7\times$ on the shorter-context datasets and approximately $22.5\times$ on the longer-context datasets. The latency benefit of block-KV reuse is therefore more pronounced on longer contexts without a corresponding increase in quality loss.

4.3.2. Direct Whole-Block KV and NLL Validation

Table 5 reports the direct whole-block validation results. Relative to Sentence-Concat, AD-Cut reduces $D_{\mathrm{KV,all}}$ by 0.0559 (paired-bootstrap 95% CI $[0.0544, 0.0575]$ in magnitude), $D_{\mathrm{KV,nfblock}}$ by 0.0509 ($[0.0495, 0.0524]$), and absolute answer-NLL mismatch by 0.3769 ($[0.3357, 0.4188]$). All four datasets show the same direction.

4.3.3. Evaluator Validation and Alias-Containment Check

A human annotator blindly evaluated 120 responses from 40 questions (10 per dataset and all three methods). GPT-5.5 and the human agreed on $113/120 = 0.9417$ responses (question-cluster bootstrap 95% CI $[0.9000, 0.9750]$), with quadratic-weighted $\kappa = 0.9167$ ($[0.8375, 0.9771]$). As a deterministic lexical complement,

normalized answer-alias containment was 0.7025 for Full Recompute, 0.5375 for Sentence-Concat, and 0.5815 for AD-Cut. A response was counted as containing an answer when at least one normalized nonempty reference alias occurred in the normalized generated response.

Table 4. Q3 Task A dataset-level comparison of online TTFT, block granularity, and judge-based QA quality for full recomputation and block-KV reuse strategies.

Dataset	Method	N	Mean Seg.	TTFT ms	TTFT Speedup	Judge Correct	Judge Score	Quality Ret.
2Wiki	Full Recompute	500	1.00	187.6	1.00×	**0.694**	**0.701**	**1.000**
2Wiki	Full Reuse Sentence-Concat	500	33.09	**24.6**	**7.62**×	0.476	0.483	0.627
2Wiki	AD-Cut Block Reuse	500	17.11	24.9	7.53×	0.506	0.513	0.667
HotpotQA	Full Recompute	500	1.00	250.8	1.00×	**0.794**	**0.820**	**1.000**
HotpotQA	Full Reuse Sentence-Concat	500	44.25	**25.7**	**9.74**×	0.630	0.643	0.715
HotpotQA	AD-Cut Block Reuse	500	23.66	26.0	9.66×	0.636	0.663	0.753
NQ	Full Recompute	500	**1.00**	668.2	1.00×	**0.680**	**0.706**	**1.000**
NQ	Full Reuse Sentence-Concat	500	111.26	**29.2**	**22.85**×	0.486	0.520	0.687
NQ	AD-Cut Block Reuse	500	65.20	29.6	22.55×	0.522	0.556	0.742
TriviaQA	Full Recompute	500	**1.00**	688.4	1.00×	**0.808**	**0.818**	**1.000**
TriviaQA	Full Reuse Sentence-Concat	500	108.56	**30.2**	**22.82**×	0.684	0.698	0.829
TriviaQA	AD-Cut Block Reuse	500	63.51	30.5	22.57×	0.750	0.760	0.891

Note: Bold values indicate the best value for each metric.

Table 5. Direct mismatch against dense full-context prefill on all 2000 frozen Task A questions. Dense prefill is the zero-difference reference.

Method	$D_{\mathrm{KV,all}}$	$D_{\mathrm{KV,nfblock}}$	ΔNLL	$\lvert\Delta\mathrm{NLL}\rvert$	Valid
Sentence-Concat	0.5661	0.5766	-1.0624	1.5217	2000
AD-Cut	0.5102	0.5257	-0.5880	1.1449	2000

4.3.4. Task B: Controlled Agent-Memory Ability

Table 6 reports the controlled agent-memory ability results. Full Recompute provides the quality baseline with a macro average of 0.2035. Full Reuse Sentence-Concat obtains 0.1638, while AD-Cut reaches 0.1847. We account for shared parent questions using category-stratified parent-cluster bootstrap. A complementary parent-weighted sensitivity analysis gives each parent equal weight within category and preserves the same qualitative method ordering, with scores of 0.2112, 0.1633, and 0.1891 for Full Recompute, Sentence-Concat, and AD-Cut, respectively.

Table 6. Q3 Task B comparison of controlled agent-memory ability scores under full recomputation and block-KV reuse. All methods are evaluated on 1000 variants derived from 433 represented parent questions in a 451-question source pool.

Method	Static Recall	Dynamic Tracking	Workflow Know.	Env. Gotchas	Premise Aware.	Avg.
Full Recompute	**0.3383**	**0.1410**	**0.2683**	**0.2000**	0.0700	**0.2035**
Full Reuse Sentence-Concat	0.2746	0.1347	0.2096	0.0650	0.1350	0.1638
AD-Cut Block Reuse	0.2779	0.1177	0.1979	0.1250	**0.2050**	0.1847

Note: Bold values indicate the best value for each metric.

AD-Cut outperforms Full Reuse Sentence-Concat on Static Recall, Environment Gotchas, and Premise Awareness. It performs lower on Dynamic Tracking and Workflow Knowledge. These differences indicate that the five memory-skill categories respond differently to block granularity.

5. Limitations

Our experiments are limited to 8B-scale models and memory contexts constrained by GPU memory and KV-cache footprint. Stability on larger models and longer memory streams remains unverified.

AD-Cut evaluates candidate boundaries independently using local incoming and outgoing signals and therefore does not guarantee global optimality under block-budget, latency, or task constraints. Future work may explore global optimization, adaptive thresholds, and task-aware boundary selection.

6. Conclusions

We introduced AD-Cut, which uses attention-derived incoming and outgoing dependency signals to partition long-term agent memory into KV blocks suitable for independent prefill and reuse. The results show that these signals are associated with intervention-derived contextual sensitivity and that, together with the structural guard, the selected boundaries provide a favorable balance between textual validity and model-side independence. In block-KV serving, AD-Cut substantially reduces Online TTFT while retaining more task quality than naive sentence-level reuse. Direct validation on 2000 questions further shows lower whole-block KV and absolute answer-NLL mismatch than sentence-level reuse. In controlled Task B, parent-aware analyses preserve the qualitative method ordering under alternative aggregation. AD-Cut shifts the focus of KV-cache reuse from how to compose cached states to where agent memory should be segmented, providing an interpretable and signal-driven approach to long-term agent-memory serving.

Author Contributions: Y.Z.: conceptualization, methodology, software, validation, formal analysis, investigation, data curation, visualization, writing—original draft preparation; B.W. and X.H.: supervision. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies: During the preparation of this work, the authors used GPT-5.5 (OpenAI) to assist with coding. After using this tool, the authors reviewed, tested, and modified the AI-assisted code as necessary and take full responsibility for the content of the published article.

Appendix A. Structural Guard Rules and Motivation

AD-Cut segments memory into sentence-level atomic units based on punctuation. However, punctuation within numbered, alphabetic, or Roman-numeral list markers, such as `1.`, `a.`, and `iv.`, can produce marker-only units or short fragments. Their short token spans may introduce attention-signal artifacts and create spurious candidate boundaries.

AD-Cut therefore applies a deterministic structural guard after the low-incoming and low-outgoing decision using only the text of adjacent atomic units. After Markdown stripping and whitespace normalization, rules are evaluated in first-match order: (i) block a standalone left list marker; (ii) block a left unit ending in a numeric, alphabetic, or Roman-numeral marker; (iii) block if either unit is empty, a standalone marker, or contains at most two whitespace-separated tokens, subject to exceptions for terminal “?”, “!”, “:”, and explicit `user:`/`assistant:` prefixes; and (iv) block a direct `user:` question to `assistant:` answer boundary.

The exact regular expressions and executable rule order are:

```
MARKDOWN = [*_`#>]+
STANDALONE = (?:[-*\u2022]|\(?\d{1,3}\)?[.)]|
              \(?[A-Za-z]\)?[.)]|\(?[ivxlcdmIVXLCDM]+\)?[.)])
ENDING = (?:^|\s)(?:\(?\d{1,3}\)?[.)]|\(?[A-Za-z]\)?[.)]|
         \(?[ivxlcdmIVXLCDM]+\)?[.)])$

APPLY_MINIMAL_GUARD(left, right, attention_candidate):
  if not attention_candidate: return NOT_A_CANDIDATE
  left, right = normalize_whitespace(left), normalize_whitespace(right)
  if standalone_list_marker(left): return BLOCK
  if left_ends_with_list_marker(left): return BLOCK
  if fragment(left) or fragment(right): return BLOCK
  if "user:" in lower(left) and "?" in left
     and lower(right).startswith("assistant:"): return BLOCK
  return KEEP_ATTENTION_CANDIDATE
```

The guard cannot create a cut or modify either attention score. Because it uses deterministic local text patterns rather than model parameters, attention values, or dataset labels, it is broadly applicable to sentence-based segmentation pipelines with similar punctuation artifacts. Deferred lexical, delimiter, abbreviation, punctuation, and

lowercase-continuation rules were not used in the reported experiments.

Appendix B. Q1 Data Construction Details

Appendix B.1. Context Construction

The formal Q1 pool contains 500 LongMemEval source samples and 9,654 unique source memory units. Each memory context is constructed by selecting up to 12 complete memory units from a single source sample with a fixed random seed. The selected units are ordered by their original source indices to preserve chronology. Memory units from different source samples are never mixed, and atomic units are not shuffled. The context token budget is $7500 \times r$, where $r \sim \mathrm{Uniform}(0.8, 1.2)$. A seeded contiguous token window is extracted only when an individual memory unit exceeds this budget. All sample IDs, memory-unit selections, and serialized contexts are frozen before the formal experiments.

Appendix C. Q2 Split and Overlap Audit

The development/test split and the five development folds were generated with master seed 20260923 using a construction-stratified, memory-unit-graph-aware procedure. Because the memory-unit graph contains a 2757-sample connected component, a strictly memory-unit-disjoint 1000/2000 split is infeasible. Relative to seeded stratification, the graph-aware procedure reduces the development/test shared-memory-unit count from 1659 to 1553.

Appendix D. Q2 Benchmark Implementations

All benchmark methods operate on the same sentence-level atomic units and select boundaries only between adjacent units.

Appendix D.1. Fixed Length

This method places target cuts at 256-token intervals and selects the atomic-unit boundary closest to each target position.

Appendix D.2. Recursive Structure

Atomic units are accumulated in their original order. When adding the next unit would cause the current segment to exceed 256 tokens, a boundary is placed after the preceding unit.

Appendix D.3. TextTiling

For each candidate boundary, TextTiling computes the lexical cosine similarity between the left and right windows using lowercase bag-of-words vectors. Depth scores are derived from local similarity valleys. The formal benchmark uses a window size of $w = 2$ and a sample-level depth percentile of $p = 0.95$.

Appendix D.4. TopicTiling

We use the official Java TopicTiling v1.0 implementation with bundled JGibbLDA: 100 topics, $\alpha = 0.5$, $\beta = 0.01$, 99 effective sampling iterations, window size 1, 100 inference iterations, five repeated inferences, and mode counting. LDA is trained on 14,221 unused raw memory units. Training and evaluation have zero memory-unit-key overlap, but all 500 source records are represented in both; this is therefore memory-unit-disjoint but not source-record-disjoint domain training. Returned segment ends are mapped deterministically to the frozen candidate boundaries.

Appendix D.5. LlamaIndex Semantic Splitter

We use `llama-index-core` 0.13.6, `SemanticSplitterNodeParser`, BAAI/bge-m3 embeddings, `buffer_size=1`, and `breakpoint_percentile_threshold=95`. A sentence callback returns the exact frozen atomic units.

Appendix D.6. Chonkie SemanticChunker

We use Chonkie 1.5.2 `SemanticChunker` with BAAI/bge-m3 embeddings, threshold 0.9, chunk size 2,048, similarity window 5, minimum three sentences per chunk, minimum 24 characters per sentence, skip window 0, filter window 7, polynomial order 3, and filter tolerance 0.2. This setting was frozen after a paired-200 development sensitivity analysis and is not presented as a package default. Chunk ends are mapped by exact offset when possible

and otherwise to the nearest frozen candidate, with the lower boundary ID breaking ties.

Appendix E. Q2 LLM Annotation Prompt and Scoring Rules

Appendix E.1. Annotation Prompt

GPT-5.5 annotated each candidate boundary without access to attention, A_{in}, A_{out}, KV, NLL, or any other hidden signals. The exact prompt was:

> You are annotating candidate cut boundaries between normalized atomic units in Agent memory. Decide whether cutting between the left and right units would improve self-contained memory chunks.
>
> must: A required cut. Not cutting would merge clearly independent memories, topics, tasks, sessions, speakers, or QA exchanges.
>
> good: A recommended cut. Cutting clearly improves chunk independence, but missing it is not catastrophic.
>
> acceptable: An optional cut. Cutting is coherent and safe, but the boundary is not semantically necessary.
>
> unacceptable: A discouraged cut. Cutting would mildly hurt local coherence, references, explanation flow, list structure, or turn continuity.
>
> bad: A clearly wrong cut. Cutting would split a list marker from its content, a short fragment, a phrase, a definition, a tightly coupled example, or a dependent QA/explanation.
>
> Prefer `bad` or `unacceptable` when the right unit depends on the left unit for syntax, reference, list membership, or explanation continuity. Prefer `must` or `good` when the boundary separates sessions, speakers, topics, tasks, independent memories, or self-contained QA turns. Use only the visible text and provide one concise reason for each label.

Appendix E.2. Scoring Rule

Selecting a boundary labeled `must`, `good`, `acceptable`, `unacceptable`, or `bad` receives a score of $+2$, $+1$, 0, -1, or -2, respectively. Leaving the same labels unselected receives a score of -2, -1, 0, 0, or 0. The raw annotation score sums these scores over all candidate boundaries. Annotation Adjusted normalizes this score against the expected random-selection score and the annotation-oracle score at the same cut count, thereby controlling for segmentation granularity.

Appendix F. Evaluator Validation Protocol

The Q2 human-validation sample contains 100 candidate boundaries, 20 from each GPT label. The Q3 sample contains 40 questions, 10 per Task A dataset, and all three anonymized method responses for each question (120 judgments). Stable SHA-256 ordering with master seed 20260922 freezes both samples. One annotator works without GPT labels, GPT reasons, attention values, method identities, or other methods' answers. We report exact agreement and quadratic-weighted Cohen's κ; these are GPT–human, not human–human, consistency measures.

Appendix G. Q3 Judge Prompt, Runtime, and Parsing

The API model identifier is `gpt-5.5`; the chat-completions API uses temperature 0, a 120-second timeout, and at most two retries. The 2000 grouped requests yield 6000 parsed method responses and no errors. The exact prompt is:

```
You are judging whether a model answer is correct for a RAG question.

Inputs:
- `question`: the question asked to the model. It may be empty for
  legacy smoke artifacts.
- `answer_aliases`: acceptable reference answers. Any alias may be
  sufficient.
- `predictions`: a list of model answers for the same question. Each
  item has `method` and `prediction`.

Judging rules:
- Mark `correct` when the prediction clearly gives or contains an
  acceptable answer alias, even if it includes extra explanation.
- Mark `correct` when the prediction is a grammatical sentence whose
```

```
  subject or answer phrase is an acceptable alias.
- Mark 'partial' only when the prediction is related but incomplete or
  ambiguous, and does not clearly contain an acceptable alias.
- Mark 'incorrect' for the wrong entity, wrong relation, contradiction,
  refusal, empty answer, or 'Unknown' when an answer alias is provided.
- Extra text is allowed unless it changes the answer or contradicts the
  alias.
- Do not require exact string match or short-answer formatting.
- Ignore capitalization, punctuation, articles, and singular/plural
  differences when they do not change meaning.

Judge each prediction independently. Return exactly one JSON object and
no other text:
{
  "results": [
    {
      "method": "method id copied from input",
      "verdict": "correct|partial|incorrect",
      "score": 1.0,
      "contains_alias": true,
      "reason": "one short sentence"
    }
  ]
}
Use 'score=1.0' for 'correct', 'score=0.5' for 'partial', and
'score=0.0' for 'incorrect'.
```

The parser requires one result per requested method and coerces only the declared verdicts to their fixed scores; invalid or missing output triggers the configured retry and is excluded only after retries are exhausted.

References

1. Wu, D.; Wang, H.; Yu, W.; et al. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. In Proceedings of the 13th International Conference on Learning Representations, Singapore, 24–28 April 2025; pp. 86809–86836.
2. Wu, D.; Ji, Z.; Kawatkar, A.; et al. LongMemEval-V2: Evaluating Long-Term Agent Memory toward Experienced Colleagues. *arXiv* **2026**, arXiv:cs.cl/2605.12493.
3. Yao, J.; Li, H.; Liu, Y.; et al. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. In Proceedings of the EuroSys'25: Twentieth European Conference on Computer Systems, Rotterdam, The Netherlands, 30 March–3 April 2025; pp. 94–109. https://doi.org/10.1145/3689031.3696098.
4. Yang, J.; Hou, B.; Wei, W.; et al. KVLink: Accelerating Large Language Models via Efficient KV Cache Reuse. In Proceedings of the 39th International Conference on Neural Information Processing Systems, San Diego, CA, USA, 2–7 December 2025.
5. Beltagy, I.; Peters, M.E.; Cohan, A. Longformer: The Long-Document Transformer. *arXiv* **2020**, arXiv:2004.05150.
6. Dao, T.; Fu, D.; Ermon, S.; et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In Proceedings of the 36th International Conference on Neural Information Processing Systems, New Orleans, LA, USA, 28 November–9 December 2022; Volume 35, pp. 16344–16359. https://doi.org/10.52202/068431-1189.
7. Hu, J.; Huang, W.; Wang, W.; et al. EPIC: Efficient Position-Independent Caching for Serving Large Language Models. In Proceedings of the 42nd International Conference on Machine Learning, Vancouver, BC, Canada, 13–19 July 2025; Volume 267, pp. 24391–24402.
8. Kwon, W.; Li, Z.; Zhuang, S.; et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. In Proceedings of the SOSP '23: 29th Symposium on Operating Systems Principles, Koblenz, Germany, 23–26 October 2023; pp. 611–626. https://doi.org/10.1145/3600006.3613165.
9. Zheng, L.; Yin, L.; Xie, Z.; et al. SGLang: Efficient Execution of Structured Language Model Programs. In Proceedings of the Advances in Neural Information Processing Systems 37, Vancouver, BC, Canada, 10–15 December 2024; pp. 62557–62583. https://doi.org/10.52202/079017-2000.
10. Gim, I.; Chen, G.; Lee, S.S.; et al. Prompt Cache: Modular Attention Reuse for Low-Latency Inference. In Proceedings of the Machine Learning and Systems, Santa Clara, CA, USA, 13–16 May 2024; Volume 6.

11. Ma, D.; Wang, Y.; Lan, T. Block-Attention for Efficient Prefilling. In Proceedings of the International Conference on Learning Representations, Singapore, 24–28 April 2025; Volume 2025, pp. 63774–63788.
12. Chen, C.; Zhang, G.L.; Yin, X.; et al. KV Packet: Recomputation-Free Context-Independent KV Caching for LLMs. *arXiv* **2026**, arXiv:cs.lg/2604.13226.
13. Zhao, J.; Ji, Z.; Fan, Z.; et al. MoC: Mixtures of Text Chunking Learners for Retrieval-Augmented Generation System. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vienna, Austria, 27 July–1 August 2025; Volume 1, pp. 5172–5189. https://doi.org/10.18653/v1/2025.acl-long.258.
14. Li, S.; Zhang, Z.; Wang, Y.; et al. Towards Generalization of Block Attention via Automatic Segmentation and Block Distillation. *arXiv* **2026**, arXiv:cs.cl/2605.15913.
15. Vaswani, A.; Shazeer, N.; Parmar, N.; et al. Attention Is All You Need. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30, pp. 5998–6008.
16. Xiao, G.; Tian, Y.; Chen, B.; et al. Efficient Streaming Language Models with Attention Sinks. In Proceedings of the Twelfth International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024; Volume 2024, pp. 21875–21895.
17. Su, J.; Lu, Y.; Pan, S.; et al. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arXiv* **2021**, arXiv:2104.09864.
18. Lambert, N.; Morrison, J.; Pyatkin, V.; et al. Tulu 3: Pushing Frontiers in Open Language Model Post-Training. *arXiv* **2024**, arXiv:2411.15124.
19. Grattafiori, A.; Dubey, A.; Jauhri, A.; et al. The Llama 3 Herd of Models. *arXiv* **2024**, arXiv:2407.21783.
20. OpenAI. GPT-5.5 Model. Available online: https://developers.openai.com/api/docs/models/gpt-5.5 (accessed on 21 August 2026).
21. Butler, U. Semchunk: A Python Library for Semantic Chunking. Available online: https://github.com/isaacus-dev/semchunk (accessed on 23 August 2026).
22. Hearst, M.A. TextTiling: Segmenting Text into Multi-Paragraph Subtopic Passages. *Comput. Linguist.* **1997**, *23*, 33–64. https://www.aclweb.org/anthology/J97-1003.pdf.
23. Riedl, M.; Biemann, C. TopicTiling: A Text Segmentation Algorithm Based on LDA. In Proceedings of the ACL 2012 Student Research Workshop, Jeju Island, Korea, 9–11 July 2012; pp. 37–42.
24. LlamaIndex. Semantic Splitter. Available online: https://docs.llamaindex.ai/en/stable/api_reference/node_parsers/semantic_splitter/ (accessed on 21 August 2026).
25. Chen, J.; Xiao, S.; Zhang, P.; et al. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings through Self-Knowledge Distillation. *arXiv* **2024**, arXiv:2402.03216.
26. Minhas, B.; Nigam, S. Chonkie: The Lightweight Ingestion Library for Fast, Efficient and Robust RAG Pipelines. Available online: https://github.com/chonkie-inc/chonkie (accessed on 23 August 2026).
27. Lewis, P.; Perez, E.; Piktus, A.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Proceedings of the Neural Information Processing Systems, Online, 6–12 December 2020; Volume 33, pp. 9459–9474.
28. Ho, X.; Duong Nguyen, A.K.; Sugawara, S.; et al. Constructing a Multi-Hop QA Dataset for Comprehensive Evaluation of Reasoning Steps. In Proceedings of the 28th International Conference on Computational Linguistics, Online, 8–13 December 2020; pp. 6609–6625. https://doi.org/10.18653/v1/2020.coling-main.580.
29. Yang, Z.; Qi, P.; Zhang, S.; et al. HotpotQA: A Dataset for Diverse, Explainable Multi-Hop Question Answering. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 31 October–4 November 2018; pp. 2369–2380. https://doi.org/10.18653/v1/d18-1259.
30. Kwiatkowski, T.; Palomaki, J.; Redfield, O.; et al. Natural Questions: A Benchmark for Question Answering Research. *Trans. Assoc. Comput. Linguist.* **2019**, *7*, 453–466. https://doi.org/10.1162/tacl_a_00276.
31. Joshi, M.; Choi, E.; Weld, D.; et al. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver, BC, Canada, 30 July–4 August 2017; pp. 1601–1611. https://doi.org/10.18653/v1/p17-1147.