TOWARDS GENERALIZATION OF BLOCK ATTENTION VIA AUTOMATIC SEGMENTATION AND BLOCK DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Block attention, which processes the input as separate blocks that cannot attend to one another, offers significant potential to improve KV cache reuse in long-context scenarios such as Retrieval-Augmented Generation (RAG). However, its broader application is hindered by two key challenges: the difficulty of segmenting input text into meaningful, self-contained blocks, and the inefficiency of existing block fine-tuning methods that risk degrading performance. To address these, we first construct SemanticSeg, a large and diverse semantic segmentation dataset containing over $30k$ instances across 16 categories—including books, code, web text, and conversations with text lengths ranging from $2k$ to $32k$. Using this dataset, we train a lightweight segmenter to automatically partition text into human-instinct-aligned blocks with controllable granularity. Second, we propose block distillation, a training framework that is more efficient than block fine-tuning, which uses a frozen full-attention teacher model to guide the block-attention student. This framework integrates three novel components: block sink tokens to mitigate information loss at block boundaries, block dropout to leverage training signals from all blocks, and token-level loss weighting to focus learning on block-attention-sensitive tokens. Experiments across multiple models and benchmarks demonstrate that our segmenter outperforms heuristic and statistical baselines, and block distillation achieves near-full-attention performance under block attention, establishing a practical and scalable pathway for deploying block attention.

1 INTRODUCTION

Large language models (LLMs) have demonstrated remarkable capabilities in processing long-context inputs (Bai et al., 2025; 2024; Maharana et al., 2024), enabling applications such as multi-document question answering, coding, etc. However, the standard full-attention mechanism scales quadratically with sequence length, making inference on long inputs computationally expensive and memory-intensive. A significant source of this inefficiency is the context-dependent nature of full attention: when identical context is paired with different prefixes, its key-value (KV) states must be recomputed from scratch. This leads to substantial waste of compute and energy, particularly in retrieval-augmented generation (RAG) scenarios where overlapping document sets are repeatedly processed across queries. To mitigate this, block attention (Ma et al., 2025) has emerged as a promising alternative. By restricting self-attention to independent blocks and allowing only a final aggregation block to attend globally, it eliminates cross-block dependencies and facilitates the reuse of pre-computed KV cache. Nevertheless, its practical adoption is hindered by several obstacles.

An important barrier in the application of block attention is segmentation, that is, how to divide the input sequence into separate blocks. Existing approaches often rely on heuristic rules, such as splitting with newlines; however, such rules rarely generalize and are highly likely to break the semantic coherence of the inputs. As demonstrated in Section 5.2, such naive segmentation leads to performance degradation, highlighting the need for a semantic-aware approach. To address this, we propose a robust, data-driven semantic segmenter capable of handling diverse input formats. We take a data-driven approach and construct a segmentation dataset (*SemanticSeg*), where each sample is segmented by the semantic meaning. Using this dataset, we train a neural segmenter that can automatically produce adaptive and context-aware boundaries, overcoming a major obstacle to the generalization of block attention.

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

Another major challenge is effectively integrating block attention into existing LLMs. While training-free strategies such as Prompt Cache (Gim et al., 2024) and Superposition prompting (Merth et al., 2024) attempt to enable KV state reuse or parallel processing, such direct application of block attention into general domains suffers from severe performance degradation compared to full attention (Table 3). This indicates that these approaches are not sufficiently effective for supporting reliable block attention, further highlighting the necessity of specialized training. (Ma et al., 2025) addressed this through "block fine-tuning", which trains models on both block and full attention patterns to balance specialized performance with general capability. However, this approach is computationally expensive and generalizes poorly across diverse domains (Section 5.2). To address these limitations, we introduce *Block Distillation*, a training framework designed for higher efficiency and signal density. It incorporates three novel mechanisms: *block sink tokens* to counteract information loss at block boundaries, *block dropout* to exploit training signals from all blocks, and *token-level loss weighting* to emphasize tokens that are most sensitive to block attention.

To validate our approach, we conduct comprehensive experiments across multiple models and benchmarks. To verify the effectiveness of our segmenter, we quantify the impact of segmentation on downstream performance and compare our segmenter against a range of heuristic and statistical segmentation baselines. To evaluate our training framework, we rigorously tested *Block Distillation*, evaluating its efficiency and performance in general domains and the specific contributions of its individual components. Experimental results demonstrate that our segmenter consistently outperforms all baselines, and *Block Distillation* pushes block-attention performance close to the full-attention upper bound while preserving or even improving full-attention capability, establishing a practical and scalable pathway for deploying block attention in long-context applications. We discuss the potential application of the block attention in Appendix D.

2 PRELIMINARY

We illustrate the idea of block attention with the example of RAG. Consider a pool of r documents and two instances with overlapping retrieved contexts:

$$\begin{aligned}&\text{Inst 1: Document } [i]; \text{ Document } [i+1]; \ldots; \text{ Document } [i+x]; \ldots; [\text{Query } q_x].\\ &\text{Inst 2: Document } [j]; \text{ Document } [j+1]; \ldots; \text{ Document } [j+y]; \ldots; [\text{Query } q_y].\end{aligned}$$

where the document sets intersect at $\{i, \ldots, i+x\} \cap \{j, \ldots, j+y\} = \{n, \ldots, m\}$. In standard full attention, the KV states for this intersection must be recomputed for each query because the attention mechanism is prefix-dependent, resulting in significant computational overhead and energy waste. However, if encoding can be performed independently for each document, the KV states for the intersection $\{n, \ldots, m\}$ become prefix-agnostic and can be safely reused across disparate queries. Block attention (Ma et al., 2025) formalizes this by partitioning the input into independent blocks. Each block employs self-attention restricted to its own tokens, ensuring that internal representations are decoupled from other blocks. Only the final block (typically the user query) is permitted to utilize full attention, aggregating information from all preceding KV caches.

The implementation of this method can be achieved easily via the following steps: 1) Independently encoding each block except the last one; 2) Computing the positional encoding for each token based on their position in the input text; 3) Concatenating all pre-computed KV states of the blocks and using them to compute the KV states for the final block. However, the previous work (Ma et al., 2025) has demonstrated that the model cannot accommodate this pattern without training, which is also verified in this work (section 5.2). To cope with this challenge, they employ block fine-tuning, which updates the parameters in both ways, one for block attention and one for full attention. They claim this could enhance block-attention performance while maintaining full attention capability. However, despite its heavy updating scheme, it struggles to generalize to other domains (5.2).

3 AUTOMATIC SEGMENTATION

A major barrier to the general application of block attention is segmentation, i.e., given the input text, how to cut it into meaningful, self-contained, and human-instinct-aligned blocks (or chunks) for later processing. One may argue that the segmentation operation has a limited effect on the final performance, as the model may not understand the semantics in the same way as humans. However, as proved in the section 5.2, the segmentation plays an important role in the final performance.

To enable general automatic segmentation, we adopt a data-driven approach and first construct a semantic segmentation dataset, called *SemanticSeg*. Then we design the segmenter and its processing approach, and use the built dataset to train it.

3.1 SEMANTICSEG

We diversify the source and length ($l \in [2k, 32k]$) of the dataset to facilitate the generalization of the segmenter. For each input text, we follow step 1 in Fig. 1 to insert candidate cut tokens and use Gemini-2.5-Pro to determine the final segmentation results. SemanticSeg contains around 16 categories of segmentation data, with each category containing at least around $2k$ instances. The varying cut rates across categories can also help the segmenter learn distinct segmentation patterns. More details of the dataset can be found in Appendix B.1.

3.2 SEGMENTATION

We construct the segmenter with a pre-trained language model backbone (Qwen3-4B-Instruct-2507) by adding a classification head consisting of two linear layers and an intermediate ReLU activation layer. The segmentation process is presented in Fig. 1. For an input text T, a set of candidate cut tokens $C \in \{C_0, C_1, \ldots, C_n\}$ is first inserted into the input text via simple rules like the newline character. This forms a series of initial text segments $\{C_0, T_1, C_1, T_2, C_2, \ldots, T_n, C_n\}$, where $\{T_1, T_2, \ldots, T_n\} = T$. The segmenter then takes this series of text segments, along with the candidate cut tokens, as input and outputs a binary probability distribution for each candidate cut token. As the current candidate cut point cannot capture important segmentation information from its successive text segments (For example, in Fig. 1, ”¡cut 3¿” cannot attend to its successive token ”Method”, which is a promising choice for final segmentation), we use the hidden vector from the next candidate to determine the segmentation of the current candidate.

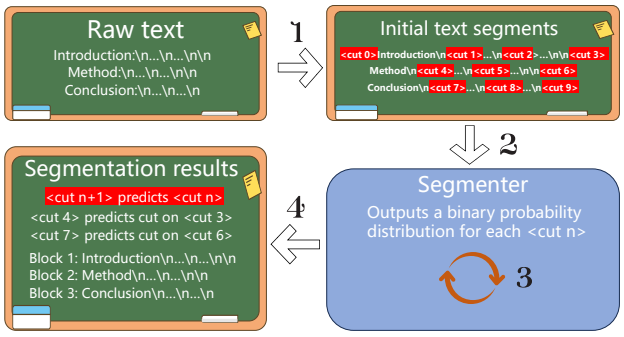

Figure 1: The segmentation process. 1. The candidate cut tokens are first inserted into the raw text via a simple rule (newline in the example). 2. The initial text segments are fed into the segmenter, which outputs a binary probability distribution for each candidate cut token. 3. The segmenter can be applied recursively with different division thresholds to customize the segmentation granularity. 4. The segmentation for each candidate cut token is determined by the corresponding consecutive cut token.

A very important factor of the segmentation is the granularity, which determines the final number of blocks (the parallel degree) and how much information each block contains. In the segmenter, two adjustable components could be used to control the segmentation granularity, which are the threshold value for the binary probability distribution and the recursion depth (the number of times the segmenter is applied recursively, with each level splitting existing blocks further using a predefined threshold) in the segmenter. The users can pair each level of recursion with a different threshold value. Generally, the deeper recursion level can pair with a greater or equal threshold value. During training, the threshold value is set to 0.5, but we recommend 0.2 ˜ 0.5 for the first level of recursion (An example is provided in Appendix B.2).

4 BLOCK DISTILLATION

Block fine-tuning (Ma et al., 2025) trains the model under block and full attention to preserve full-attention performance. This is time-consuming and hard to scale. Therefore, we propose block distillation, which uses the original full-attention model to guide block-attention training. With block distillation, we can safely bypass the heavy updating scheme of block fine-tuning without degrading full-attention performance while improving block-attention performance (section 5.2).

The block distillation employs three novel components to facilitate the training. They are *block sink tokens* that are used to mitigate abnormal patterns in block attention, *block dropout* that takes advantage of the non-last block training signal, and *token weighting* applied to the token dimension to the cross-entropy loss.

4.1 Block Sink Tokens

The previous investigation (Zhang et al., 2025) reveals that the attention patterns are extremely abnormal at the beginning of each block, leading to potential optimization instabilities. We characterize this challenge as *lost in block head* (section 5.3). To tackle this problem, we introduce a new token ($< |block_start| >$) called block sink token. Following (Xiao et al., 2024), we duplicate the block sink token four times at the beginning of each block. Hence, the final block-attention version input follows the format $\{bls * 4, B_1, bls * 4, B_2, \ldots, bls * 4, B_n\}$, where bls means the block sink token $< |block_start| >$ and $B_r, r \in [1, n]$ is the blocks partitioned by the segmenter. In practice, we set the dropout rate to around 0.6 for all the training.

4.2 Block Dropout

A fundamental requirement for block attention is the model's ability to accurately retrieve information from the KV caches of all the blocks. Existing fine-tuning methods (Ma et al., 2025) are highly inefficient because they only optimize the model using signals from the final block, essentially ignoring the vast majority of the input sequence. To address this signal sparsity problem, we introduce block dropout (Fig. 2). This mechanism randomly selects a subset of context blocks for individual encoding (blue in Fig. 2) and applies a KL divergence loss to all remaining non-corrupted blocks (orange). By doing so, we force the model to learn from a much larger proportion of the text. Formally, given an input sequence x, a frozen teacher model φ, a student model φ_s, and let $\mathfrak{R}(x)$ denote the set of tokens within corrupted blocks. The block dropout KL divergence is defined as:

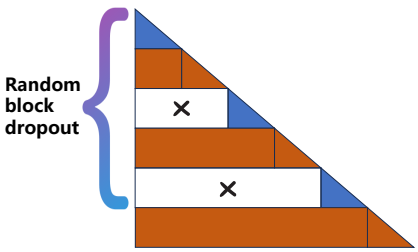

Figure 2: The block dropout. A number of randomly selected blocks are forced to attend only the content within the block itself. Note that the final block always follows the full-attention pattern.

$$KL_x = D_{KL}(p_\varphi(\{x_i | x_i \notin \mathfrak{R}(x)\}) \,||\, p_{\varphi_s}(\{x_i | x_i \notin \mathfrak{R}(x)\})) \tag{1}$$

4.3 Token Weighting

Traditional cross-entropy loss applies equal weights to the token dimension. Such a mechanism potentially decreases the training effectiveness, as it contains no information on the different degrees of importance for each token. Thus, we introduce the token weights that are employed on the token dimension of the cross-entropy computation, similar to (Li et al., 2025). Specifically, given a teacher model φ whose weights are frozen, an input x, the token weights are computed as follows:

$$w_x = max(CE(\varphi_b(x)) - CE(\varphi(x)), 0) \times \alpha + \beta \tag{2}$$

where φ_b means a block-attention forward pass in the teacher model and CE is the cross-entropy loss. The token weights assign greater weight to tokens that have a relatively large difference in loss between the block-attention and full-attention forward passes, and shrink the loss scale of those insensitive tokens ($CE(\varphi_b(x)) - CE(\varphi(x) \leq 0$) to β (usually a value near 0.1). In this way, we can alleviate the noise in training and focus more on the learning of block-attention capability. We set $\alpha = 0.2, \beta = 0.1$ for Qwen series models and $\alpha = 0.5, \beta = 0.1$ for Llama series models in the later experiments.

4.4 Training

The final training loss is the combination of the previously introduced components. Specifically,

$$loss_x = CE(\varphi_{bs}(x)) \times w_x + KL_x \tag{3}$$

where φ_{bs} represents a block-attention forward pass in the student model. More training details have been put into Appendix C.2.

5 EXPERIMENTS

In this section, we conduct comprehensive experiments to verify the key arguments of this paper from two perspectives. From the perspective of the segmentation: 1) The degree to which segmentation influences downstream performance, and 2) whether our segmenter offers a genuine improvement over other straightforward partition methods. From the perspectives of block distillation: 1) Whether block fine-tuning is enough for the general domain application, 2) whether the block distillation can help block-attention performance approach that of full attention in the general domain, 3) whether the block distillation affect the full attention performance, 4) what is the efficiency gain brought by the block attention in inference, and 5) the effectiveness of each component in block distillation.

5.1 EXPERIMENT SETTINGS

Benchmarks We adopt two popular comprehensive benchmarks for evaluation, namely LongBench (Bai et al., 2024) and LoCoMo (Maharana et al., 2024), and follow the exact procedures defined in the original papers, including the metrics, prompts, etc.

Baselines Unlike (Ma et al., 2025), which needs to reproduce the whole SFT procedure, we implement the block distillation directly on chat models, including Qwen3-4B-Instruct-2507, Llama-3.1-8B-instruct, Qwen3-8B, and Qwen3-14B. For fair comparison, we also include the model from previous work (Ma et al., 2025). The details for the baselines are as follows:

- **Original** - The untouched official model released on HuggingFace. This is the performance upper bound for the block attention model. Our goal is to make the block attention model's overall performance as close as possible to this model's.
- **Block-Dist** - The model trained via our block distillation framework using our segmented data processed by the trained segmenter.
- **Prompt Cache** - The Prompt Cache (Gim et al., 2024) baseline, which enables training-free reuse of the attention states across prompts.
- **Superposition** - The Superposition Prompting baseline (Merth et al., 2024). It allows LLMs to process the input documents in parallel paths, and prunes the irrelevant paths.
- **Tulu3-SFT** - The original Llama-3.1-Tulu-3-8B-SFT model, which serves as the ceiling performance for the Tulu3 series of block attention baselines from (Ma et al., 2025).
- **Tulu3-Block-FT** - The block attention model trained by block fine-tuning (Ma et al., 2025), using the SFT dataset of Tulu3 and 20,000 samples of RAG data sampled from TriviaQA and 2WikiMultiHopQA. We include this model to visualize the gap between its block-attention performance and Tulu3-SFT's full-attention performance in the general domain.
- **Tulu3-Block-FT-S** - Since the training data of Tulu3-Block-FT is partitioned by simple rules without the segmenter, we further train Tulu3-Block-FT using our data divided by the segmenter for fair comparison.

Unless otherwise specified, "- Full" indicates that the test is under full attention, and "- Block" means evaluation is using block attention.

5.2 MAIN RESULTS

The impact of segmentation In this section, we verify two points: How much impact does the segmentation have on the performance, and is the segmenter really better than other simple segmentation baselines? These two points are verified by comparing the segmenter with other segmentation methods (Table 1). Specifically, we include two sets of baselines, one of which is segmentation in *heuristics*:

- *Random* - Segmentation in random with the set of candidate cut points to be the space.
- *Average* - Segmentation in average with the set of candidate cut points to be the space.
- *Punctuation* - Use sentence-ending punctuation as candidate cut points for segmentation.

Method	Multi-doc QA	Single-doc QA	Summarization	Few shot	Synthetic	Code	Average
Qwen3-8B							
Full	40.52	47.51	24.01	35.59	66.99	01.66	37.85
Random	20.53	36.77	22.96	10.51	08.69	04.73	17.37
Average	19.78	36.66	22.83	11.10	07.91	05.02	17.22
Punctuation	06.14	11.62	21.55	08.72	02.85	06.25	09.52
Random candidate	18.72	35.34	22.99	22.29	02.44	07.49	18.21
Average candidate	17.24	33.11	22.83	21.86	02.32	07.01	17.40
Loss	17.66	27.98	22.34	19.74	03.44	08.58	18.02
Entropy	17.29	28.18	22.58	16.88	03.09	08.31	17.41
Segmenter - 4B	17.88	36.92	22.81	24.05	03.65	04.61	18.32
Segmenter - 0.6B	17.53	35.85	23.14	23.41	03.54	04.59	18.01
Tulu3-Block-FT							
Tulu3-SFT - Full	42.09	45.03	25.24	38.20	65.67	24.61	40.40
Random	36.96	40.49	15.30	27.13	19.57	29.89	30.38
Average	36.05	40.52	24.75	28.95	15.35	27.66	28.88
Punctuation	10.76	13.27	22.24	19.39	08.52	17.47	15.28
Random candidate	36.84	40.40	25.17	45.00	03.12	30.74	30.21
Average candidate	35.58	39.82	25.02	45.24	03.49	30.16	31.12
Loss	32.66	34.94	24.67	36.46	05.41	28.49	28.01
Entropy	30.31	33.76	24.59	35.87	3.83	32.40	27.35
Segmenter - 4B	34.33	42.93	24.93	45.34	12.17	31.28	32.82
Segmenter - 0.6B	33.51	42.37	23.99	45.11	12.89	31.78	31.61

Table 1: Results for different segmentation methods on Longbench (Bai et al., 2024). For fair comparison, the parallel degree for all methods is aligned with that of the segmenter.

Method	Multi-doc QA	Single-doc QA	Summarization	Few shot	Synthetic	Code	Average
Llama-3.1-Tulu-3-8B-SFT							
Tulu3-SFT - Full	42.09	45.03	25.24	38.20	65.67	24.61	40.40
Tulu3-Block-FT - Block	34.33	42.93	24.93	45.34	12.17	31.28	32.82
Tulu3-Block-FT-S - Block	52.11	31.93	24.43	36.23	15.12	39.21	33.17

Table 2: Main results on Block-FT. " - Full" means the evaluation is performed using full attention, and "- Block" means the evaluation is performed under block attention.

- *Random candidate* - Random segmentation using the user-predefined candidate cut points.
- *Average candidate* - Average segmentation using the user-predefined candidate cut points.

Another set is segmentation in *statistics*:

- *Loss* - The segmentation with the chunked topk cross-entropy loss value[1].
- *Entropy* - The segmentation with the chunked topk token entropy value.
- *Segmenter* - The segmentation using the segmenter with different size (Qwen3-4B-instruct-2507 and Qwen3-0.6B) trained by *SemanticSeg*.

To exclude the impact of segmentation during training, we do not use models trained via block distillation. Instead, we apply these methods on two models, namely, Qwen3-8B - the original chat model, and Tulu3-Block-FT - the block attention model trained via block fine-tuning (Ma et al., 2025). The parallel degree of all segmentation methods is aligned with that of the segmenter.

The performance variance of different segmentation baselines from Tulu3-Block-FT is noticeable, demonstrating the impact of segmentation methods. Since Qwen3-8B is not trained on any segmented data, the performance variance of Qwen3-8B is generally lower than that of Tulu3-Block-FT. Although these two models do not use any training data partitioned by the segmenter, they both perform the best on average when the input is processed by the segmenter (in comparison with other segmentation baselines[2]). This significantly demonstrates the effectiveness of the segmenter. In ad-

[1]We use the chunked topk to prevent cut points from clustering together

[2]Note that the random candidate and the average candidate generally follow the segmentation methods used by Tulu3-Block-FT.

Method	Multi-doc QA	Single-doc QA	Summarization	Few shot	Synthetic	Code	Average
Qwen3-4B-Instruct-2507							
Original - Full	36.88	46.05	22.08	61.63	66.27	04.89	39.63
Original - Prompt Cache	28.36	31.89	17.89	49.23	48.49	04.83	30.12
Original - Superposition	25.39	34.85	18.38	41.52	45.72	02.32	28.02
Block-Dist - Full	38.74	47.14	22.33	59.22	67.66	12.70	41.29
Block-Dist - Block	37.55	46.17	24.14	59.00	66.32	11.98	40.86
Llama-3.1-8B-Instruct							
Original - Full	41.75	47.05	25.86	24.16	65.91	15.77	36.75
Original - Prompt Cache	33.28	39.87	22.41	19.45	50.76	14.83	30.10
Original - Superposition	31.95	37.81	25.75	21.45	46.15	13.87	29.50
Block-Dist - Full	42.00	47.42	25.88	25.67	66.50	20.11	37.93
Block-Dist - Block	41.88	46.98	25.35	24.96	66.87	11.89	36.32
Qwen3-8B							
Original - Full	40.52	47.51	24.01	35.59	66.99	01.66	36.05
Original - Prompt Cache	28.89	38.86	23.77	20.76	45.27	05.67	27.20
Original - Superposition	29.75	35.86	28.96	22.78	49.65	08.51	29.25
Block-Dist - Full	42.75	47.11	25.75	38.97	67.33	11.33	38.87
Block-Dist - Block	41.77	46.38	25.17	36.76	67.00	09.95	37.83
Qwen3-14B							
Original - Full	46.25	48.86	23.94	62.44	69.17	05.66	42.72
Original - Prompt Cache	32.48	35.80	24.58	52.83	49.42	08.64	33.96
Original - Superposition	35.88	32.85	22.77	48.54	45.23	10.41	32.61
Block-Dist - Full	47.23	47.75	22.87	63.45	70.05	17.71	44.84
Block-Dist - Block	46.33	46.54	22.89	63.76	69.23	06.43	42.53

Table 3: Main results on LongBench (Bai et al., 2024).

dition, the minor performance gap between the two segmenters validates the possibility of further decreasing the segmenter size and hence lowering the cost of segmentation.

Generalization failure of block FT The previous work (Ma et al., 2025) only tests the block attention under the RAG scenario. Hence, how the block fine-tuning performs for the general domain remains an unverified point. Therefore, we first test the block attention model trained in the previous work (Ma et al., 2025) to find out whether it can achieve similar performance to the full attention model. The results are shown in Table 2. The Tulu3-Block-FT model has degraded block attention performance compared to that of the Tulu3-SFT - Full. There are two possibilities for this: one is the incompetence of the block fine-tuning, and the other is the difference in the training data that is segmented via simple rules. To eliminate the influence of the training data, we further train the "Tulu3-Block-FT" model using our training data partitioned by our segmenter, the model denoted as "Tulu3-Block-FT-S". The results show that the block-attention performance of this model ("Tulu3-Block-FT-S - Block") has a relatively noticeable gap compared to the full-attention performance of "Tulu3-SFT". Therefore, we can conclude that the source of the performance gap is the block fine-tuning, and it is not enough for the generalization of block attention.

Effectiveness of block distillation In this section, we verify the effectiveness of block distillation in both block attention and full attention. The results are shown in Table 3 and Table 4. The Prompt Cache (Gim et al., 2024) and Superposition prompting (Gim et al., 2024) considerably degrade the model performance compared to vanilla full-attention. This aligns with the findings in (Ma et al., 2025) and section 5.2 that the model struggles to adapt the block attention pattern without training. For all models, both the block-attention and full-attention performance of block distillation achieves near-equivalent performance to that of full-attention from the original model. Therefore, the block distillation can help improve the block-attention capability of the model while preserving its full-attention ability.

Efficiency We measure both training and inference efficiency (Table 6). For training, block distillation requires $25,859.9ms$ per step, which is approximately 26% faster than Block-FT ($34,941.1ms$), which demonstrates the efficiency of block distillation over block fine-tuning. For inference, we follow the input format $[context, user\ query]$ and measure the time-to-first-token (TTFT) for vanilla full-attention and block attention across context lengths from $8k$ to $64k$. The

Method	Multi hop	Single hop	Temporal	Open domain	Adversarial	Average
Qwen3-4B-Instruct-2507						
Original - Full	39.98	33.38	09.60	48.83	34.75	33.31
Original - Prompt Cache	19.78	20.24	05.76	31.57	19.64	19.40
Original - Superposition	17.38	19.51	07.42	28.40	22.85	19.11
Block-Dist - Full	38.86	34.09	09.93	49.34	32.34	32.91
Block-Dist - Block	37.05	32.67	12.60	48.57	32.02	32.58
Llama-3.1-8B-Instruct						
Original - Full	49.76	38.67	18.65	66.15	16.82	38.01
Original - Prompt Cache	32.61	25.75	10.75	50.65	15.37	27.03
Original - Superposition	25.51	28.64	15.51	40.48	17.42	25.51
Block-Dist - Full	50.25	36.57	17.82	67.72	17.23	37.92
Block-Dist - Block	48.55	36.82	19.79	65.22	20.73	38.22
Qwen3-8B						
Original - Full	40.44	32.43	18.28	58.30	23.32	34.55
Original - Prompt Cache	29.71	25.62	15.37	48.27	17.51	27.30
Original - Superposition	20.51	21.41	11.24	39.39	19.48	22.41
Block-Dist - Full	39.08	34.03	19.87	56.28	23.99	34.65
Block-Dist - Block	38.76	32.71	16.26	57.38	22.49	33.52
Qwen3-14B						
Original - Full	38.95	33.16	14.26	60.45	37.44	36.85
Original - Prompt Cache	29.68	28.87	07.34	51.36	22.97	28.04
Original - Superposition	27.62	20.63	04.39	42.59	19.41	22.93
Block-Dist - Full	39.93	34.93	13.06	61.10	36.61	37.13
Block-Dist - Block	37.95	33.82	14.73	59.89	37.30	36.74

Table 4: Main results on LoCoMo (Maharana et al., 2024).

Method	Multi-doc QA	Single-doc QA	Summarization	Few shot	Synthetic	Code	Average
Qwen3-4B-Instruct-2507							
Block-Dist - Block	37.55	46.17	24.14	59.00	66.32	11.98	40.86
- w/o Block sink tokens	36.19	44.86	22.56	55.38	52.33	17.96	38.21
- w/o Block dropout	36.21	45.22	23.71	45.35	43.23	13.23	34.49
- w/o KL loss	38.26	38.22	21.56	39.67	40.87	14.25	32.13
- w/o Token weights	42.87	39.87	23.78	59.29	45.50	20.99	38.71

Table 5: The ablation study results.

length of the user query is set to 200. Block attention consistently achieves lower TTFT than vanilla full-attention, and the absolute gain grows with sequence length: the TTFT reduction increases from $57.9ms$ at $8k$ to $3,149.7ms$ at $64k$. We use Qwen3-8B and GPU H20 for all the efficiency testing.

5.3 Ablation Study & Analysis

The lost in block head Given such a segmented example from the Longbench (Bai et al., 2024) synthetic task:

> "Block 1 - Paragraph 1:...; Block 2 - Paragraph 2:...; Block 3 - Paragraph 3:...;...; Block n - The following is an abstract:..., Please enter the number of the paragraph that the abstract is from. The answer format must be like "Paragraph 1", "Paragraph 2", etc. The answer is: "

where the model is required to retrieve the information from the head (beginning) of the block. We find that the block attention model has serious trouble in dealing with this type of query. We name this phenomenon *lost in block head*. We believe that the source of the problem is linked to the findings in a previous investigation (Zhang et al., 2025), which shows that the attention pattern is

extremely abnormal at the head of the blocks. Interestingly, despite the enormously greater amount of training data used compared to this work, the block-FT model trained in (Ma et al., 2025) shows a considerable collapse in this synthetic task (Tulu3-Block-FT - Block in Table 3), indicating that simple fine-tuning may not be enough.

Method	Time (ms)
Training	
Block-FT	34,941.1
Block-Dist	25,859.9
Inference	
TTFT-vanilla - $8k$	509.2
TTFT-block - $8k$	451.3
TTFT-vanilla - $16k$	1,099.5
TTFT-block - $16k$	702.1
TTFT-vanilla - $32k$	2,642.5
TTFT-block - $32k$	1,534.8
TTFT-vanilla - $64k$	6,971.3
TTFT-block - $64k$	3,821.6

Table 6: The efficiency measurement.

Block sink tokens To tackle the lost in block head problem, we introduce a new special token "$<|block_start|>$", which is padded before each block's beginning to allieviate the abnormal attention pattern. We conduct an experiment to verify its effectiveness in Table 5. The performance shows a considerable decrease in the Synthetic task, which aligns with the discussion in section 5.3. In addition, the few-shot and single-doc QA tasks also experience a noticeable drop, implying that the block sink token not only helps the understanding of the information in the block head but also the later actual block content.

Block dropout We verify the effectiveness of the block dropout and KL divergence on the Longbench benchmark (Bai et al., 2024) (Table 5) via forcing computing the KL loss for the last block only. The results demonstrate a great decrease in the few-shot and synthetic tasks when the block dropout is absent during training, suggesting it helps alleviate the lost in block head problem. The KL loss component has also been verified by completely wiping it out. The results experience a tremendous drop in single-doc QA, few-shot, and synthetic tasks, suggesting its important role in helping learn block-attention patterns.

Token weights The effectiveness of the token weights is verified by using the usual mean reduction for the cross-entropy loss. Although adopting the cross-entropy without weights increases the Multi-doc QA performance, it contrarily degrades the performance in single-doc QA and synthetic tasks.

6 Related work

Block attention (Ma et al., 2025) partitions input sequences into independent blocks to enable efficient KV cache reuse and parallel prefilling, but its broader adoption is hindered by costly block fine-tuning and limited generalization beyond RAG settings. Prompt Cache (Gim et al., 2024) explores training-free modular attention reuse across prompts. Superposition prompting (Merth et al., 2024) processes documents in parallel paths and prunes irrelevant ones. CacheBlend (Yao et al., 2025), KVLink (Yang et al., 2025) and SpecCache (Wen et al., 2026) reuse non-prefix KV cache with selective KV recompute. Unlike these methods that adopt the full prefix KV cache, C^2KV (Du et al., 2026) loads the compressed KV and concatenates it with the input. AdapShot (Ou et al., 2026) adopts semantic KV cache reuse to accelerate LLM inference in many-shot In-Context Learning scenario. However, the existing approaches suffer from severe performance degradation when directly applied under block-attention patterns without dedicated training. In contrast, our work introduces a learned semantic segmenter and an efficient block distillation framework that overcomes these limitations, achieving block-attention performance close to the full-attention upper bound while preserving full-attention capability.

7 Conclusion

In this work, we address two fundamental obstacles that prevent the broader adoption of block attention: the absence of a principled segmentation method and the inefficiency of existing block fine-tuning. To tackle the first, we construct SemanticSeg, a large-scale multi-domain segmentation dataset, and train a lightweight neural segmenter that partitions text into semantically coherent blocks with controllable granularity. To overcome the second, we propose Block Distillation, an efficient training framework that incorporates three novel components—block sink tokens, block dropout, and token-level loss weighting—to effectively transfer full-attention capability to the block-attention pattern. Extensive experiments on LongBench and LoCoMo across multiple model families demonstrate the effectiveness of the segmenter and Block Distillation.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish the writing.

ETHICS STATEMENT

By boosting long-context efficiency and KV cache reuse, our work lowers compute cost and energy use, making advanced AI more accessible and sustainable for coding assistants, multi-document analysis, and agentic applications. However, cheaper inference may lower barriers for misuse (e.g., disinformation) and accelerate deployment without labor safeguards. Efficiency benefits may not transfer equally across architectures or languages, risking a widening gap between well-supported and underrepresented settings. Responsible deployment and monitoring are encouraged.

REPRODUCIBILITY STATEMENT

The datasets (SemanticSeg), code and trained models (the Segmenter and block attention models) will be released.

REFERENCES

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 3119–3137. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.172. URL https://doi.org/10.18653/v1/2024.acl-long.172.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 3639–3664. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.183/.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=6PmJoRfdaK.

Alexis Chevalier, Jiayi Geng, Alexander Wettig, Howard Chen, Sebastian Mizera, Toni Annala, Max Jameson Aragon, Arturo Rodríguez Fanlo, Simon Frieder, Simon Machado, Akshara Prabhakar, Ellie Thieu, Jiachen T. Wang, Zirui Wang, Xindi Wu, Mengzhou Xia, Wenhan Xia, Jiatong Yu, Junjie Zhu, Zhiyong Jason Ren, Sanjeev Arora, and Danqi Chen. Language models as science tutors. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 8310–8335. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/chevalier24a.html.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels. *CoRR*, abs/2412.05496, 2024. doi: 10.48550/ARXIV.2412.05496. URL https://doi.org/10.48550/arXiv.2412.05496.

Chuheng Du, Junyi Chen, Hanlin Tang, Kan Liu, Tao Lan, Lin Qu, Chaoyue Niu, Shengzhong Liu, Guihai Chen, and Fan Wu. C^2kv: Compressed and composable KV cache reuse for efficient LLM inference. In Won Kim, Jae-Gil Lee, Kyuseok Shim, James Caverlee, David F. Gleich, Xiangliang Zhang, Qi He, Faisal Farooq, Rómer Rosales, Sayan Ranu, Haixun Wang, Yan Liu,

Carl Yang, Leman Akoglu, Nitesh V. Chawla, and Vagelis Papalexakis (eds.), *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2, KDD 2026, Jeju Island, Korea, August 9-13, 2026*, pp. 991–1002. ACM, 2026. doi: 10.1145/3770855.3817715. URL https://doi.org/10.1145/3770855.3817715.

In Gim, Guojun Chen, Seung-Seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa (eds.), *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. Liger kernel: Efficient triton kernels for LLM training. *CoRR*, abs/2410.10989, 2024. doi: 10.48550/ARXIV.2410.10989. URL https://doi.org/10.48550/arXiv.2410.10989.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 TB of permissively licensed source code. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=pxpbTdUEpD.

Wojciech Kryscinski, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. BOOKSUM: A collection of datasets for long-form narrative summarization. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Findings of ACL, pp. 6536–6558. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.FINDINGS-EMNLP.488. URL https://doi.org/10.18653/v1/2022.findings-emnlp.488.

Shuaiyi Li, Zhisong Zhang, Yang Deng, Chenlong Deng, Tianqing Fang, Hongming Zhang, Haitao Mi, Dong Yu, and Wai Lam. Incomes: Integrating compression and selection mechanisms into llms for efficient model editing. *CoRR*, abs/2505.22156, 2025. doi: 10.48550/ARXIV.2505.22156. URL https://doi.org/10.48550/arXiv.2505.22156.

Xiaoyuan Liu, Tian Liang, Dongyang Ma, Deyu Zhou, Haitao Mi, Pinjia He, and Yan Wang. The pensieve paradigm: Stateful language models mastering their own context. *CoRR*, abs/2602.12108, 2026. doi: 10.48550/ARXIV.2602.12108. URL https://doi.org/10.48550/arXiv.2602.12108.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Dongyang Ma, Yan Wang, and Tian Lan. Block-attention for efficient prefilling. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=7zNYY1E2fq.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 13851–13870. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.747. URL https://doi.org/10.18653/v1/2024.acl-long.747.

Thomas Merth, Qichen Fu, Mohammad Rastegari, and Mahyar Najibi. Superposition prompting: Improving and accelerating retrieval-augmented generation. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 35507–35527. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/merth24a.html.

Jie Ou, Jinyu Guo, Shiyao Guo, Yuang Li, Ruiqi Wu, Zhaokun Wang, Wenyi Li, and Wenhong Tian. Adapshot: Adaptive many-shot in-context learning with semantic-aware KV cache reuse. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 42946–42960. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.1990. URL https://doi.org/10.18653/v1/2026.acl-long.1990.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=jKHmjlpViu.

Zhiqiang Shen, Tianhua Tao, Liqun Ma, Willie Neiswanger, Zhengzhong Liu, Hongyi Wang, Bowen Tan, Joel Hestness, Natalia Vassilieva, Daria Soboleva, and Eric P. Xing. Slimpajama-dc: Understanding data combinations for LLM training. *CoRR*, abs/2309.10818, 2023. doi: 10.48550/ARXIV.2309.10818. URL https://doi.org/10.48550/arXiv.2309.10818.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 9835 musique: Multihop questions via single-hop question composition. *Trans. Assoc. Comput. Linguistics*, 10:539–554, 2022. doi: 10.1162/TACL_A_00475. URL https://doi.org/10.1162/tacl_a_00475.

Zijian Wen, Tao Zhang, Shuangwu Chen, Shenghao Ye, Yu Guo, Qirui Chen, Jingxian Shuai, Yunpeng Hou, Huasen He, and Jian Yang. Speccache: Speculative KV cache reuse for efficient RAG serving. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 18861–18871. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.859. URL https://doi.org/10.18653/v1/2026.acl-long.859.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=pZiyCaVuti.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Peng Xu, Wei Ping, Xianchao Wu, Chejian Xu, Zihan Liu, Mohammad Shoeybi, and Bryan Catanzaro. Chatqa 2: Bridging the gap to proprietary llms in long context and RAG capabilities. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=cPD2hU35x3.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. Kvlink: Accelerating large language models via efficient KV cache reuse. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/c217d123885341853cecbdd2be809983-Abstract-Conference.html.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 2369–2380. Association for Computational

Linguistics, 2018. doi: 10.18653/V1/D18-1259. URL https://doi.org/10.18653/v1/d18-1259.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pp. 94–109. ACM, 2025. doi: 10.1145/3689031.3696098. URL https://doi.org/10.1145/3689031.3696098.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhisong Zhang, Yan Wang, Xinting Huang, Tianqing Fang, Hongming Zhang, Chenlong Deng, Shuaiyi Li, and Dong Yu. Attention entropy is a key factor: An analysis of parallel context encoding with full-attention-based pre-trained language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 9840–9855. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.485/.

A LIMITATIONS

There are several limitations of this work that we need to discuss.

Block dropout in pretraining So far, we have only explored employing the block dropout in the post-training phase. However, we believe it is possible to scale it to the pre-training phase to further improve the capability. At that time, the cross-entropy loss can also be applied to the non-corrupted block.

Thinking mode verification We do not verify the thinking mode function of the block attention model since the training data does not contain any thinking content. However, based on the investigation from previous work (Zhang et al., 2025), the thinking mode has high potential for being effective in block attention.

Model type & size Due to resource limitations, we can only scale the model size to 14B. The effectiveness of block distillation to block attention under a larger model size requires further exploration. Additionally, the experiments are only conducted for dense models; the compatibility of block attention with other popular structures, like MoE, requires further discussion.

RL compatibility Reinforcement learning is a powerful tool for improving the model's reasoning and agentic capabilities nowadays. However, none of the existing works investigates the compatibility of block attention with RL. Therefore, we think this is a promising direction for further exploration. If they are proven compatible, then the cost of commercial models would be considerably reduced.

Agentic application This paper does not verify the effectiveness of block attention in agentic scenarios. However, a big advantage of block attention is the KV cache reuse across prompts. Considering the complicated context engineering operations for agents these days (Liu et al., 2026), if it can be applied to agentic applications such as multi-agent collaboration, it could save much waste on re-encoding, and the cost would be massively decreased.

Category	Source	Num	Cut rate
Book chapter	Booksum (Kryscinski et al., 2022)	3551	0.0851
Long instruction	LongAlphaca (Chen et al., 2024)	3895	0.0724
Short Paragraphs	MuSiQue (Trivedi et al., 2022)	3254	0.9260
Chat history	LongMemEval (Wu et al., 2025)	3100	0.1315
Textbook chapter	TextbookChapters (Chevalier et al., 2024)	1980	0.1031
Mathematical text	OpenWebMath (Paster et al., 2024)	1980	0.1259
ArXiv	SlimPajama (Shen et al., 2023)	1980	0.0268
Raw book	SlimPajama (Shen et al., 2023)	1980	0.0313
StackExchange QA	SlimPajama (Shen et al., 2023)	1980	0.0251
Educational web pages	FineWeb-Edu (Lozhkov et al., 2024)	1980	0.1157
Wikipedia	SlimPajama (Shen et al., 2023)	1980	0.1015
Code - Comprehensive	The stack (Kocetkov et al., 2023)	4821	0.2022
Code - Python	The stack (Kocetkov et al., 2023)	1980	0.1190
Code - C	The stack (Kocetkov et al., 2023)	1980	0.1227
Code - Java	The stack (Kocetkov et al., 2023)	1980	0.1125
Code - Shell	The stack (Kocetkov et al., 2023)	1980	0.1783

Table 7: The statistics for the SemanticSeg dataset. "Comprehensive" means all the existing code categories in The stack (Kocetkov et al., 2023). The cut rate is calculated as the number of authenticated cut tokens ÷ the number of candidate cut tokens.

B More details about segmentation

B.1 SemanticSeg Dataset

The specific statistics and the resources for *SemanticSeg* dataset are shown in Table 7. The prompt used for generating the segmentation data is as follows:

Prompt used by Gemini-2.5-Pro for *SemanticSeg*

```
You are a master editor specializing in text segmentation for
parallel processing by Large Language Models.  Your goal is
to segment a given text into the largest possible, yet fully
self-contained, semantic chunks.  The text is pre-segmented with
candidate markers `<cut 0>` to `<cut N>`.


Your task is to identify the correct boundaries by selecting
a subset of these markers.  A chunk is "self-contained" if a
person (or an LLM) can understand it completely in isolation,
without needing to read the preceding chunk.


**The Core Principle:** The ideal cutting point is where the
topic COMPLETELY changes, and the new chunk can be understood
without any ambiguity.  If a chunk starts with a sentence that
makes you ask "wait, who/what/why are they talking about?", then
the cut is wrong and the chunks must be merged.


**CRITICAL HEURISTICS FOR COHERENCE (Rules for what NOT to
do):** You must merge chunks avoid these common errors.  A cut
is BAD if:

1.  **It splits a sentence.** The text after a "<cut *>" marker
cannot be the grammatical continuation of a sentence from before
the marker.  Example:  "...he demanded", "Do the Maquas dare..."
-> BAD CUT. Merge the two chunks to form a complete sentence.
```

```
2.  **It breaks direct anaphora (reference).** The new chunk
cannot start with words or phrases that directly refer to
the immediate preceding text.  Examples:  "I'll try those
suggestions.", "Because of this...", "That's a great point.",
"He/she then...", "As they..." -> BAD CUT. The reference ("those
suggestions", "this", "That", "He/She", "They", etc.)  is in the
previous chunk.  Merge the reference chunk with its immediately
previous chunk.

3.  **It separates a question from its answer, or a statement
from its immediate response.** Dialogue is a continuous flow.  A
multi-turn exchange on a single, evolving topic should be in ONE
chunk.  Example:  A user asks for coffee shops, the assistant
lists them, the user asks about Wi-Fi at one of them.  This
entire thread about finding a coffee shop is ONE semantic unit
and should be in the SAME chunk.

4.  **It separates functionally dependent content.** Text that
references an element (like a figure or table) must be in
the same chunk as that element's description.  Example:  Text
discussing "Figure 1" must be in the same chunk as the caption
for "Figure 1".

5.  **It separates the chunk content from its header.** The
chunk content and its corresponding header should always stay
in the same chunk.  Example:  "Chapter V\n", "Mary did something
bad...\n" -> BAD CUT. The header ("Chapter", "Section", "Part",
etc.)  and its content should be in the SAME chunk.  Merge the
header chunk with its content chunk.

6.  **It breaks the semantic clarity.** Each chunk should
remain understandable to a downstream model or retrieval system.
Examples:  For Python, "def hello_word(a:str)\n", " print("hello
world!")" -> BAD CUT. The function head and its content should
not be separated; "class myclass()\n", " def __init__(self,
config):" -> BAD CUT. The class head and its content should not
be separated.  Merge them.


**Cutting steps you should refer:** 1.  Read the entire text
to get a general sense of its structure (narrative, dialogue,
academic paper, code, etc.).  2.  Iterate through each candidate
marker from `<cut 1>` to `<cut N-1>`.  3.  For each marker,
examine the two chunks immediately before and after it.  Ask
yourself whether this marker violates the heuristics above and
whether it fulfills the core principle above.  4.  If the marker
violates any of the above heuristics or does not fulfill the
above core principle, identify this as a BAD cut point, move
to the next marker and repeat step 3.  Otherwise, identify this
marker as a GOOD cut point.  5.  Based on the identified GOOD
cut points, construct your final output.


**NOTE:** (1).  Usually, the final number of chunks should be
greater than or equal to 5.  (2).  For the length of each chunk,
it should contain no more than 350 candidate markers.  (3).  Try
your best to fulfill both (1) and (2).  If you can't, you can
compromise between (1) and (2) depend on the circumstances.
```

```
(4). Extremely large chunk is not allowed! Putting extremely
large content into a chunk (For example, putting a whole paper
into a chunk) to avoid BAD cuts is unacceptable!


**Output Format:** Follow these rules exactly: 1. Chunk
boundaries must sit only on the exact strings `<cut *>`. 2.
Output **only** lines of the form and **nothing else**: `chunk
<number>: <cut *> --- <cut **>` where `<cut *>` and `<cut
**>` are the beginning and ending cutting point markers of the
current chunk. 3. Number chunks sequentially starting with 1.
4. The beginning marker of the first chunk must be `<cut 0>`
and the final marker of the final chunk must be `<cut N>`. 5.
Do not output any duplicated chunks.


Text to segment:
```

B.2 Segmentation Example

We provide a segmentation example about python code to facilitate the understanding. The example uses recursion depth 1 and threshold value 0.4.

An example processed by the segmenter

```
chunk 1:
import numpy as np
from numpy.testing import assert_array_equal,
assert_array_almost_equal
import scipy.stats.distributions as distrs
from scipy.stats.kde import gaussian_kde
from scipy.integrate import quad
import pytest

chunk 2:
def augment_grid(x, n_inner_points): test_arr = [
np.linspace(x[i], x[i + 1], n_inner_points + 1, endpoint=False)
for i in np.arange(len(x) - 1) ] test_arr.append([x[-1]]) return
np.concatenate(test_arr)
def circle_fun(x, low, high): x = np.array(x) center = 0.5 *
(high + low) radius = 0.5 * (high - low)
res = np.zeros_like(x)
center_dist = np.abs(x - center) is_in = center_dist <= radius
res[is_in] = np.sqrt(radius ** 2 - center_dist[is_in] ** 2)
return res

chunk 3:
class TestCont: """Regression tests for `Cont` class"""
def test_init_errors(self): def check_one_input(def_args,
var): with pytest.raises(TypeError, match=f"`var`.*numpy
array"): def_args[var] = "a": None Cont(**def_args) with
pytest.raises(TypeError, match=f"`var`.*float"): def_args[var]
= ["a", "a"] Cont(**def_args) with pytest.raises(TypeError,
match=f"`var`.*finite values"): def_args[var] = [0,
np.nan] Cont(**def_args) with pytest.raises(TypeError,
match=f"`var`.*finite values"): def_args[var] = [0,
np.inf] Cont(**def_args) with pytest.raises(ValueError,
match=f"`var`.*1d array"): def_args[var] = [[0, 1]]
Cont(**def_args)
```

```
check_one_input("y":  [1, 1], "x") check_one_input("x":  [0, 1],
"y")
with pytest.raises(ValueError, match="[Ll]engths.*match"):
Cont([0, 1], [1, 1, 1])
with pytest.raises(ValueError, match="two"):  Cont([1], [1])
with pytest.warns(UserWarning, match="`x`.*not sorted.*`x` and
`y`"):  rv = Cont([1, 0], [0, 2]) rv_ref = Cont([0, 1], [2, 0])
_test_equal_rand(rv, rv_ref)
with pytest.raises(ValueError, match="`y`.*negative"):  Cont([0,
1], [1, -1])
with pytest.raises(ValueError, match="`y`.*no positive"):
Cont([0, 1], [0, 0])
chunk 4:
def test_init(self):  x_ref = np.array([0, 1, 2]) y_ref =
np.array([0, 1, 0]) rv_ref = Cont(x_ref, y_ref)

chunk 5:
class TestFromRVAccuracy:  """Accuracy of `Cont.from_rv()`"""
# Output of `from_rv()` should have CDF that differs from
original CDF by # no more than `thres` pytest.mark.slow
pytest.mark.parametrize( "distr_dict,thres", [
(DISTRIBUTIONS_COMMON, 1e-4), (DISTRIBUTIONS_INF_DENSITY,
1e-3), (DISTRIBUTIONS_HEAVY_TAILS, 5e-3), ], ) def
test_cdf_maxerror(self, distr_dict, thres):  test_passed =  name:
TestFromRVAccuracy.from_rv_cdf_maxerror(distr) <= thres for name,
distr in distr_dict.items()
assert all(test_passed.values())

chunk 6:
class TestFromSampleAccuracy:  """Accuracy of
`Cont.from_sample()`"""
# Output of `from_sample()` should differ from original
density estimate by # no more than `thres` (with default
density estimator) pytest.mark.slow pytest.mark.parametrize(
"distr_dict,thres", [ (DISTRIBUTIONS_COMMON, 1e-4),
(DISTRIBUTIONS_INF_DENSITY, 1.5e-4), (DISTRIBUTIONS_HEAVY_TAILS,
1e-4), ], ) def test_close_cdf(self, distr_dict, thres):
rng = np.random.default_rng(101) test_passed =  name:
TestFromSampleAccuracy.simulated_cdf_error(distr, rng) <= thres
for name, distr in distr_dict.items()
```

The segmenter successfully recognizes the boundaries of the classes (chunk 3-6) included in the python code and it also identify those non-class boundaries (chunk 1-2), demonstrating its generalization capability.

C TRAINING DETAILS

C.1 SEGMENTER

We use all categories from the *SemanticSeg* for the training of the segmenter. Note that for the code category, we only use the comprehensive subset. The segmenter structure is an autoregressive model backbone with a new cut head. The cut head consists of two linear layers and an intermediate ReLU activation layer. We use the learning rate $2e^{-5} - 2e^{-6}$ with a cosine decay strategy.

C.2 BLOCK DISTILLATION

We use the flex attention (Dong et al., 2024) framework and liger-kernel (Hsu et al., 2024) to implement the block attention during training. For all models, we adopt a learning rate $2e^{-6} - 2e^{-7}$ and a cosine decay strategy.

For the training dataset, we adopt HotpotQA (Yang et al., 2018) and a subset from ChatQA2 (Xu et al., 2025). We use the trained segmenter to divide the samples from the subset of ChatQA2 (called ChatQA2Seg) for the training dataset. Overall, the number of training data is around $180k$, and the length is less than $32k$.

D APPLICATION SCENARIO ANALYSIS

D.1 CODING AGENT

Consider an LLM-powered coding assistant managing a large repository. A developer first asks Query A about python1.py and python2.py, and later asks Query B about python3.py and python4.py. Both queries share no overlapping files except the project's global config.

Under full attention, the KV cache is context-dependent: it is tied to the exact sequence of documents in a specific prompt. If the assistant encodes the retrieved files for Query A, the cached KV states are affected by the specific file order and Query A's tokens. To serve Query B with a different file subset or order, the cache cannot be partially reused; the entire prompt, including potentially many unchanged files, must be re-encoded. This leads to significant and wasteful recomputation.

Block attention decouples this dependency by treating each document as an independently encoded block. The KV states of python1.py, python2.py, python3.py, python4.py, etc., are stored separately. When Query B arrives, only the required blocks are fetched and composed with the new query block — no re-encoding of unchanged documents is needed. This modular, prompt-level KV cache reuse is the fundamental efficiency gain of block attention: it replaces rigid, monolithic caching with flexible, composable caching, dramatically reducing redundant prefilling in dynamic, long-context scenarios typical of coding agents and multi-document workflows.

Assume the input obeys the following format $[system\,prompt, retrieved\,code\,files, user\,query]$ and the length of the system prompt is around $15k$, the repositories contains around 30 files with each file has around $10k$ length, the user query length is 200, and we retrieve 10 files for every user query. For vanilla prefix cache, only the system prompt can be reused, hence the cache hit rate $15k \div (15k + 100k + 200) \approx 0.13$. However, for block attention, since the code files are all encoded and cache separately, they can be reused directly when retrieved. The cache hit rate $(15k + 100k) \div (15k + 100k + 200) \approx 0.99$. Given that the price with cache hit could be 10 times lower than that of the missed hit, this can save tremendous computation time and cost.

D.2 MULTI-TURN AGENTIC WORKFLOWS

Consider an LLM-based research agent tasked with "Investigate recent advances in mechanistic interpretability and summarize key findings." The agent operates in a multi-turn ReAct-style loop (Yao et al., 2023): at each step, it decides which tool to use (search, browse, or code execution), processes the results, and plans the next action. The specific turns are as follows:

Turn 1: Search for "mechanistic interpretability survey 2025" and retrieve Paper A, Paper B, and Paper C.
Turn 2: Browse Paper A and extract the main research landscape.
Turn 3: Browse Paper B and understand the evidence and limitations.
Turn 4: Browse Paper C and record its experimental design and conclusions.
Turn 5: Revisit Paper A to reuse its taxonomy for structuring the final report.
Turn 6: Revisit Paper B to compare its evidence with the results in Paper C.
Turn 7: Extract key sentences from A, B, and C to build a comparison table.
Turn 8: Compose the final report with specific citations to all three papers.

Under full attention, each new turn concatenates the growing history with newly retrieved documents and the agent's next action. Even if papers A, B, and C remain identical across turns, their KV states

are embedded in a monolithic context that changes with every search result and reasoning step. Their cached states from previous turns are rarely reusable because the surrounding prompt context and document ordering have shifted, forcing repeated re-encoding of the same documents.

With block attention, each permanent element (e.g., the system prompt, Paper A, Paper B, Paper C, etc) is encoded as an independent block and cached once. Each agent turn only requires encoding the new query, any new search results, and the fresh reasoning step, while the static document blocks are fetched directly from the cache and combined in a modular way. Over many turns and many documents, this eliminates repeated prefilling of unchanged content, yielding compounding efficiency savings and substantially lowering latency and compute cost in long-running agentic workflows.

Suppose all paper A, B and C has $20k$ in length. For vanilla prefix cache, the cached states from previous turns can only be reused as a whole. However, for block attention, since the paper A, B and C are encoded independently, their states can be safely reused once they are prefilled in turn 2-4. Therefore, the number of cache hit tokens from turn 5-7 would have a increase of $20k \times 6 = 120k$.