CROSS-FAMILY SPECULATIVE PREFILL: TRAINING-FREE LONG-CONTEXT COMPRESSION WITH SMALL DRAFT MODELS

Shubhangi Upasani[1*], **Ravi Shanker Raju**[1], **Bo Li**[1], **Mengmeng Ji**[1], **John Long**[1], **Chen Wu**[1]
Urmish Thakker[1], Guangtao Wang[2]
[1]SambaNova AI [2]Meta Inc

ABSTRACT

Prompt length is a major bottleneck in agentic large language model (LLM) workloads, where repeated inference steps and multi-call loops incur substantial prefill cost. Recent work on speculative prefill demonstrates that attention based token importance estimation can enable training-free prompt compression, but this assumes the existence of a draft model that shares the same tokenizer as the target model. In practice however, agentic pipelines frequently employ models without any smaller in-family draft model. In this work, we study cross-family speculative prefill, where a lightweight draft model from one model family is used to perform prompt compression for a target model from a different family. Using the same speculative prefill mechanism as prior work, we evaluate a range of cross-family draft–target combinations, including Qwen, LLaMA, and DeepSeek models.

Across a broad diversity of tasks, we find that attention-based token importance estimation transfers reliably across different model families despite differences in model architectures and tokenizers between draft and target models. Cross-model prompt compression largely retains 90–100% of full-prompt baseline performance and, in some cases, slightly improves accuracy due to denoising effects, while delivering substantial reductions in time-to-first-token (TTFT). These results suggest that speculative prefill depends mainly on task priors and semantic structure, thus serving as a generalizable prompt compression primitive. We discuss the implications of our findings for agentic systems, where repeated long-context inference and heterogeneous model stacks make cross-model prompt compression both necessary and practical.

1 INTRODUCTION

In agentic workloads, the performance of large language models (LLMs) is increasingly limited by the cost of prompt processing. Applications involving document-level parsing, tool use and code debugging have to make calls to a large language model i.e. the target model with long context that include reasoning traces, retrieved documents and execution logs. In these settings, the cost of prefill stage scales with prompt length and tends to become the primary bottleneck in terms of latency and throughput.

Prompt compression therefore, is an attractive avenue to reduce LLM inference cost. Recent work on Speculative Prefill (Liu et al., 2025) addresses this problem by estimating token-level importance scores using a lightweight draft model. In this setting, the target model is the large language model used to perform the final task inference on the compressed input prompt while the draft model is a smaller model used at inference time to estimate token importance to guide prompt compression. Both models belong to the same model family. In practice however, some frontier models don't have smaller in-family draft models available (eg: DeepSeek-V3.1/R1, Kimi-K2 etc (dee, 2025; Guo et al., 2025; Team et al., 2025)). Moreover, agentic systems frequently combine heterogeneous models due to cost, availability, and deployment constraints. This raises a natural question: *does attention-based token importance generalize across model families?*

*Corresponding author: `shubhangi.upasani@gmail.com`

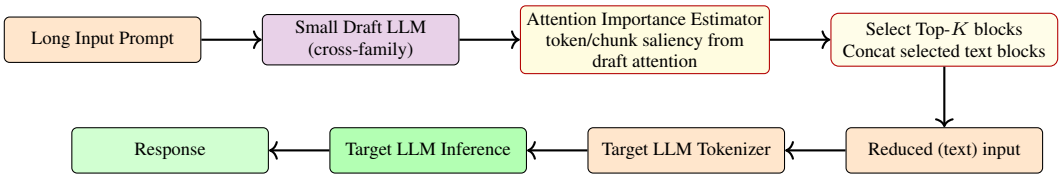

Figure 1: Cross-model speculative prefill prompt compression pipeline

In this work, we show that attention-based speculative prefill generalizes across model families, and it can be used as a reliable prompt compression mechanism for target models which don't have a supporting in-family draft model. Importantly, we do not modify the original speculative prefill algorithm itself: we use the same attention-based token scoring and chunk selection as prior work (Liu et al., 2025). The only difference is that the draft and target models are drawn from different model families, including combinations of Qwen, LLaMA, and DeepSeek models (Yang et al., 2025; Grattafiori et al., 2024; Guo et al., 2025; Liu et al., 2024).

We summarize our key contributions below:

- We present a simple, generalizable cross model approach to speculative prefill, showing that attention-driven token importance is largely transferable across model families.
- We evaluate our method on long context benchmarks like LongBench, RULER along with some code debugging tasks. Across a wide range of tasks, cross-model prompt compression largely retains 90–100% of full-prompt baseline performance, despite substantial differences in model architecture and tokenizers between draft and target models(Bai et al., 2024; 2025; Hsieh et al., 2024; Zhang et al., 2024b).
- We demonstrate up to a $\sim$18$\times$ reduction in time-to-first-token (TTFT), and show that cross-family prompt compression enables the practical use of long-context inputs under deployment constraints, by leveraging long-context draft models to compress inputs into a fixed deployable context budget for the target model.
- Our findings have direct implications for agentic workloads, where prompt length can be long. Our method enables training-free, accuracy-preserving prompt compression even when the draft model is chosen purely for cost or availability reasons.

2 Background

Inference-time prompt compression techniques reduce the length of the input prompt at inference time, without modifying model parameters or requiring additional training. Existing approaches to inference-time prompt compression can be broadly categorized by how the prompt is reduced. Rewriting-based methods such as LLMLingua (Jiang et al., 2023) employ auxiliary models to remove less informative portions of the prompt, achieving substantial prompt compression. A second class of approaches performs token or chunk-level selection, directly retaining a subset of tokens from the original prompt while preserving their original ordering. Recent work on attention-based token selection, most notably speculative prefill (Liu et al., 2025), instantiates this paradigm by using internal model signals to identify salient tokens at inference time.

Alongside prompt compression, a large body of work focuses on orthogonal inference optimizations that do not modify the input prompt itself. Attention acceleration techniques such as FlashAttention, sparse attention mechanisms (e.g., MInference, DuoAttention, LongFormer), and KV cache management strategies (e.g., H2O, SnapKV, SageKV, StreamingLLM) improve inference efficiency by reducing computation or memory usage during prefill and decoding respectively (Xiao et al., 2024a; Jiang et al., 2024; Dao et al., 2022; Beltagy et al., 2020; Zhang et al., 2023; Li et al., 2024; Wang et al., 2025; Xiao et al., 2024b). While complementary to prompt compression, these methods operate after the prompt has already been ingested by the model.

Algorithm 1 Cross-Family Speculative Prefill (Prompt Compression)

Require: Target model T, draft model D, prompt text x, lookahead steps N, keep rate ρ, chunk size c, delimiter token $\langle\text{DELIM}\rangle$
Ensure: Target output y

$\mathcal{A} \leftarrow \emptyset$
$t \leftarrow \textsc{Tokenizer}(D, x)$ ▷ Draft-tokenize; length S
for $i = 1$ to N **do**
  $(\text{attn}_i, _) \leftarrow \textsc{ForwardWithAttn}(D, t, \text{lookahead} = i)$
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{\text{attn}_i\}$
end for ▷ $\text{attn}_i \in \mathbb{R}^{L \times H \times S}$
$I \leftarrow \textsc{AggregateMaxMean}(\mathcal{A})$ ▷ $I \in \mathbb{R}^S$
$\tilde{I} \leftarrow \textsc{AvgPool1D}(I)$
$\mathcal{C} \leftarrow \textsc{PartitionIntoChunks}(\tilde{I}, c)$
$\mathcal{J} \leftarrow \textsc{TopKChunks}(\mathcal{C}, \rho)$
$\mathcal{S} \leftarrow \textsc{MergeAdjacentChunks}(\{\mathcal{C}_j : j \in \mathcal{J}\})$
$\mathcal{M} \leftarrow \textsc{MapDraftChunksToTextSpans}(t, \mathcal{S})$
$x_{\text{cmp}} \leftarrow \textsc{ConcatWithDelim}(\mathcal{M}, \langle\text{DELIM}\rangle)$
$u \leftarrow \textsc{Tokenizer}(T, x_{\text{cmp}})$ ▷ Target-tokenize
$p \leftarrow (0, 1, \ldots, |u| - 1)$ ▷ New contiguous pos-ids
$y \leftarrow \textsc{Forward}(T, u, p)$
return y

3 METHOD

We adopt the speculative prefill architecture and token selection mechanism introduced in prior work, using a lightweight draft model from a different model family to identify salient tokens in the input prompt as shown in Figure 1. Our method follows speculative prefill's core algorithmic steps but with a small but important modification to support heterogeneous draft–target model pairs. We use the same attention-based token importance estimation and chunk-based selection procedure as prior work, and defer details to Appendix A.1. Below, we describe the modifications required to enable cross-family compatibility.

3.1 KEEP RATE SELECTION

We define the keep rate ρ with respect to the target model's tokenization. Given an original target-side prompt length L_{target} and a desired compressed length $\hat{L}_{\text{target}}$, we set $\rho = \hat{L}_{\text{target}}/L_{\text{target}}$. This rate is applied in the draft domain by selecting the top-K chunks corresponding to a fraction ρ of the draft sequence based on attention-derived importance scores. The selected spans are converted back to text and re-tokenized using the target tokenizer to form the compressed prompt. Due to cross-family tokenization differences, the resulting target-side length may deviate slightly from $\hat{L}_{\text{target}}$, but is typically close in practice. Since L_{target} varies across inputs, ρ is computed per sample, and we report its average value for each benchmark.

For most of our results reported in Section 5, we select keep rates that correspond to compressed prompt lengths of 8k, 16k, 24k, and 32k tokens, which are compatible with the paged-attention execution scheme on our custom hardware Reconfigurable Dataflow Unit (RDU). In particular, KV caches are stored in 4k-token blocks, and attention computation operates over these blocks during inference. Choosing compressed prompt lengths that are multiples of this block size ensures efficient KV cache paging and streaming during attention computation.

3.2 CROSS-FAMILY DRAFT–TARGET COMPATIBILITY

After chunk selection in the draft domain, adjacent chunks are merged into contiguous spans and concatenated at the text level to construct the compressed prompt. Task-specific delimiter tokens are inserted between non-contiguous spans to explicitly mark discontinuities. Unlike the original speculative prefill formulation, which restores original positional indices, we assign new, contiguous position IDs before passing the compressed prompt to the target model (refer Algorithm 1). This avoids the need for position alignment across heterogeneous tokenization schemes, and we find that it has negligible impact on task accuracy across all evaluated benchmarks.

4 EXPERIMENTAL SETUP

4.1 BENCHMARKS

We evaluate cross-family speculative prefill on a suite of long-context benchmarks covering diverse input structures and reasoning requirements, including LongBench v1 and v2, RULER, and Code Debug task from InfiniteBench (Bai et al., 2024; 2025; Hsieh et al., 2024; Zhang et al., 2024a). LongBench v1 and v2 assess multi-document reasoning, retrieval, summarization, and code understanding across varying context lengths, while RULER and InfiniteBench stress extreme long-context settings such as needle-in-a-haystack retrieval and repository-scale code debugging. We present additional details in Appendix A.2.

4.2 MODELS AND DRAFT–TARGET PAIRINGS

We evaluate cross-family speculative prefill using a diverse set of heterogeneous target–draft model pairings spanning the Qwen, LLaMA, and DeepSeek model families. On LongBench v1 and v2, we use mid-scale target models (e.g., Qwen-8B and LLaMA-8B) paired with smaller draft models, and additionally evaluate DeepSeek-R1 as a target on LongBench v2. To study extreme long-context settings, we use RULER with DeepSeek-V3 as the target model. For code debugging tasks, we evaluate both DeepSeek-R1 and DeepSeek-V3.1 as target models. For each pairing, we explore a range of keep rates and report accuracy relative to the corresponding full-prompt baseline. Keep rates reported in the results denote average values across each benchmark, while the target-side compressed context length is held fixed for all samples within a given target–draft pairing. Additional implementation details are provided in Appendix A.3.

5 RESULTS

5.1 CROSS-FAMILY SPECULATIVE PREFILL PRESERVES TASK PERFORMANCE

Results for LongBench v2 and Code Debug are presented in Tables 1 and 2, with additional results on LongBench v1 and RULER reported in Tables 3 and 4 respectively. We use greedy decoding for all reported results. For Code Debug, the average input length for DeepSeek-R1 is $\sim$110k tokens, and keep rates of 30%, 20%, and 15% reduce the effective context to roughly 33k, 22k, and 16k tokens respectively. For LongBench v2, the average input length is 242k, 260k and 248k tokens under LLaMA-8B, Qwen3-8B and DeepSeek-R1 as target models, and the compressed lengths follow the keep rates specified in Table 1.

Across diverse benchmarks, cross-model speculative prefill maintains performance close to the full-prompt baseline (typically within 90–100%), even at low keep rates of 6% on selected Longbench V2 tasks. This robustness holds despite differences in architecture and tokenization between the draft and target model families, suggesting that token importance estimates are not strongly tied to a specific model family. For LLaMA-8B, we evaluate both an in-family draft (LLaMA-1B) and a cross-family draft (Qwen3-1.7B). The results show comparable performance across keep rates (e.g., 29.4% vs 29.6% at 25% keep rate, and 30.6% vs 31.2% at 50%). Similar trends are observed for Qwen3-8B when paired with LLaMA-1B and Qwen3-1.7B drafts. These results indicate that speculative prefill is not strongly dependent on cross-family pairing, and that small draft models of comparable size provide similar token saliency signals for prompt compression.

For DeepSeek-R1, we additionally explore keep rates of 3%, 6%, and 10% on LongBench-v2. These correspond to compressed prompt lengths of approximately 8k, 16k, and 24k tokens, respectively. We observe that performance improves as the keep rate increases from 3% to 6%, indicating the expected trade-off between compression and task performance. The accuracy decreases slightly at 10%, suggesting a diminishing return from retaining additional context. Finally, we note that the larger draft model (Qwen3-4B-Instruct-2507) achieves higher performance on DeepSeek-R1 because it supports a 262k native context length, compared to 32k for Qwen3-1.7B. The longer context allows the draft model to estimate attention over long inputs more reliably, which improves token selection during compression.

We see some degradation on Code Debug under aggressive compression (e.g., 15% keep rate), which is discussed in Appendix A.5. In some cases, most notably on LongBench v2, compressed prompts

Table 1: Cross-model speculative prefill on LongBench V2 (**SP** = speculative prefill)

Method	Keep rate	easy	hard	short	medium	long	Accuracy (%)
Target – Llama-3.1-8B-Instruct							
Full Prompt	–	31.2	31.2	36.1	29.3	26.9	31.2
SP w/ Llama-3.2-1B-Ins	25%	30.2	28.9	39.4	23.7	24.1	29.4
SP w/ Qwen3-1.7B	25%	32.8	27.7	38.3	26.5	21.3	29.6
SP w/ Llama-3.2-1B-Ins	50%	32.8	29.3	40	26.5	23.1	30.6
SP w/ Qwen3-1.7B	50%	31.8	30.9	37.8	28.4	25.9	31.2
SP w/ Qwen3-0.6B	10%	33.3	28.0	35.6	28.8	23.1	30.0
Target – Qwen3-8B							
Full Prompt	–	30.7	28.3	39.4	23.7	23.1	29.2
SP w/ Qwen3-1.7B	25%	35.4	29.6	37.8	27.9	29.6	31.8
SP w/ Llama-3.2-1B-Ins	25%	32.3	26.0	34.4	24.2	26.9	28.4
SP w/ Qwen3-1.7B	50%	33.9	26.7	38.9	22.3	27.8	29.4
SP w/ Llama-3.2-1B-Ins	50%	35.9	26.4	39.4	25.1	24.1	30.0
Target – DeepSeek-R1							
Full Prompt	–	66.1	53.4	62.2	54.4	59.3	58.3
SP w/ Qwen3-4B-Ins-2507	6%	57.8	50.5	64.4	45.1	50.9	53.3
SP w/ Llama-3.1-8B-Ins	6%	55.2	53.4	60.6	45.6	60.2	54.1
SP w/ Qwen3-1.7B	3%	50	43.4	50.6	38.1	53.7	45.9
SP w/ Qwen3-1.7B	6%	54.2	44.1	56.1	39.5	50.9	47.9
SP w/ Qwen3-1.7B	10%	53.6	42.8	58.3	37.7	46.3	46.9

Table 2: Cross-model speculative prefill on Code Debug

Method	Keep rate	Accuracy (%)
Target – DeepSeek-V3.1		
Full Prompt	–	67.51
SP w/ Llama-3.1-8B-Ins	20%	64.72
SP w/ Llama-3.1-8B-Ins	15%	59.13
Target – DeepSeek-R1-0528		
Full Prompt	–	74.37
SP w/ Llama-3.1-8B-Ins	30%	70.30
SP w/ Llama-3.1-8B-Ins	25%	68.02
SP w/ Llama-3.1-8B-Ins	15%	62.44

achieve performance slightly exceeding the full-prompt baseline (Table 1). We attribute this to the denoising effect of context compression: by removing large amounts of irrelevant context, speculative prefill simplifies the problem.

For RULER with input context lengths of 128k (Table 4), we note that the full-prompt baseline is evaluated using SnapStream KV cache compression (Li et al., 2025) on our custom RDU hardware. This is needed to make 128k-token inference feasible within on-chip memory constraints for DeepSeek-V3. This cache compression contributes to loss of long range information needed for retrieval-heavy tasks in RULER, thus observing the lower baseline performance. In contrast, speculative prefill reduces the effective context length prior to inference, thus proactively removing noisy context and mitigating this degradation. Together, these results highlight that cross model speculative prefill can preserve accuracy and in some cases, improve task performance under practical deployment constraints. Overall, cross-model speculative prefill enables substantial prompt compression while largely preserving downstream task performance. Additional details are given in A.4.

5.2 Cross model prompt compression reduces long context cost

On RULER (Table 4), cross-model speculative prefill yields substantial latency improvements by aggressively compressing long inputs. Compressing 128k token prompts to 16k using a lightweight draft model reduces TTFT from 46 seconds to approximately 2.5 seconds, corresponding to a $\sim 18\times$ reduction. Even at a more conservative compression target of 32k tokens, TTFT is reduced to 4.3 seconds. We measure these latencies on our custom RDU hardware using the same TTFT measurement methodology commonly used for GPU-based systems.

Table 3: Cross-model speculative prefill results on LongBench V1

Method	Keep rate	hotpotQA	pass.ret.en	qmsum	mulfqa.en	lcc	rep-p	mulfqa.zh	pass.ret.zh	Accuracy (%)
Target - Llama3.1-8B-Instruct										
Full Prompt	-	55.97	99.5	25.38	55.8	44.56	43.5	62.45	97.38	55.31
Spec prefill with Qwen3-1.7B	45%	55.99	100	24.9	54.32	41.09	41.33	62.4	95.25	54.29
Target-Qwen3-8B										
Full Prompt	-	59.47	100	24.03	53.84	50.49	51.13	62.45	98.5	57.34
Spec prefill with Llama-3.2-1B-Instruct	45%	59.63	100	24.05	52.56	46.21	49.74	59.69	99	55.98

Table 4: Cross-model speculative prefill results on RULER

Method	Keep rate	niah_single_1	niah_single_2	niah_single_3	niah_multikey_1	niah_multikey_2	niah_multikey_3	niah_mutlivalue	niah_multiquery	qa_1	qa_2	Accuracy (%)
Target - DeepSeek V3												
Full Prompt	-	100	100	62.8	89.2	99.6	32.2	98.75	99.25	61.2	55	80
Spec prefill with Qwen3-4B-Ins-2507	12.5%	100	100	100	100	83.4	76.4	99.6	100	70.46	66.84	89.67
Spec prefill with Llama-3.1-8B-Instruct	12.5%	100	100	100	99.8	99.6	96.8	99.2	100	70.74	67.43	93.36

5.3 Robustness to Draft Model Selection

We next evaluate the robustness of prompt compression to the choice of draft model while holding the target model fixed. Results on LongBench v2 (Table 1) indicate that effective compression can be achieved using draft models with substantially different parameter scales. For instance, with LLaMA-8B as the target model, both Qwen3-1.7B and Qwen3-0.6B produce compressed prompts that preserve performance close to the full-prompt baseline. Likewise for DeepSeek-R1, competitive results are obtained using either Qwen3-4B or LLaMA-8B as draft models. Our experiments show that small draft models (1B–1.7B range) already provide sufficiently reliable saliency estimates for 8B target models. Larger draft models (e.g., Qwen3-4B) improve performance primarily when processing very long inputs, where stronger long-context modeling leads to more accurate attention estimation. Overall, performance varies only modestly across draft model choices, suggesting that the compression quality is not strongly dependent on a specific architecture or model scale beyond a minimal effectiveness threshold. Tokens that are important for reasoning (numbers, entities, key instructions) tend to attract attention consistently across different LLM architectures trained on similar data. This robustness enables flexible draft model selection based on computational cost or resource constraints without sacrificing downstream accuracy.

5.4 Decoupling Target Context Length from Deployment Limits

Although recent open-source models such as DeepSeek-V3.1/R1 support contexts up to 128k tokens, deploying them at full length is often infeasible due to memory and hardware constraints, resulting in much smaller usable context windows in practice. For our RDU, we were limited to a maximum target side context length of 32k tokens for these models. Cross-family speculative prefill enables long-context capability by using small draft models with native long-context support (e.g., LLaMA-8B and Qwen3-4B) to process long prompts and compress them via attention-based token importance before forwarding them to these target models within a 32k-token deployment budget. This allows target models to effectively operate on information from 128k-token inputs while maintaining performance close to the full-prompt baseline on Code Debug (Tables 2). More generally, the same mechanism enables scalable context extension: a target model with limited context capacity (e.g., 128k) can leverage a longer-context draft model (e.g., 256k) to compress longer inputs and simulate full-context inference beyond its native deployment limit.

6 Conclusion

Prior speculative prefill assumes in-family drafts; we show this assumption is unnecessary. We studied cross-family speculative prefill, where a lightweight draft model from one model family estimates attention-based token importance to compress prompts for a target model from a different family. Without modifying speculative prefill's core algorithm, we support heterogeneous draft–target pairs. Across LongBench, RULER, and code debug, our results show that attention-derived importance signals are largely transferable across model families with some task-specific constraints (Appendix A.5), enabling substantial prompt compression while preserving most of the full prompt accuracy and lending significant latency benefits through TTFT reduction. Crucially, this flexibility enables practical long-context deployment under real-world hardware constraints, by allowing any available long-context draft model to compress inputs for a constrained target model. Our results have direct implications for agentic workloads where prompt processing is a major bottleneck.

REFERENCES

Deepseek-v3.1 release. DeepSeek API Documentation, Aug 2025. https://api-docs.deepseek.com/news/news250821.

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, 2024. URL https://arxiv.org/abs/2308.14508.

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, 2025. URL https://arxiv.org/abs/2412.15204.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe

Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion, 2023. URL https://arxiv.org/abs/2306.14893.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou,

Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models, 2023. URL https://arxiv.org/abs/2310.05736.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention, 2024. URL https://arxiv.org/abs/2407.02490.

Jonathan Li, Nasim Farahini, Evgenii Iuliugin, Magnus Vesterlund, Christian Häggström, Guangtao Wang, Shubhangi Upasani, Ayush Sachdeva, Rui Li, Faline Fu, et al. Snapstream: Efficient long sequence decoding on dataflow accelerators. *arXiv preprint arXiv:2511.03092*, 2025.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation, 2024. URL https://arxiv.org/abs/2404.14469.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Jingyu Liu, Beidi Chen, and Ce Zhang. Speculative prefill: Turbocharging ttft with lightweight and training-free token importance estimation, 2025. URL https://arxiv.org/abs/2502.02789.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems, 2023. URL https://arxiv.org/abs/2306.03091.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Guangtao Wang, Shubhangi Upasani, Chen Wu, Darshan Gandhi, Jonathan Li, Changran Hu, Bo Li, and Urmish Thakker. Llms know what to drop: Self-attention guided kv cache eviction for efficient long-context inference, 2025. URL https://arxiv.org/abs/2503.08879.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads, 2024a. URL https://arxiv.org/abs/2410.10819.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024b. URL https://arxiv.org/abs/2309.17453.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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, 2018. URL https://arxiv.org/abs/1809.09600.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. 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)*, pp. 15262–15277, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.814.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, et al. Infinitybench: Extending long-context evaluation beyond 100k tokens. *arXiv preprint arXiv:2402.13718*, 2024b.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. Qmsum: A new benchmark for query-based multi-domain meeting summarization, 2021. URL https://arxiv.org/abs/2104.05938.

A Appendix

A.1 Token Importance Estimation and Chunk Selection

A.1.1 Attention-Based Token Importance Estimation

Given an input prompt of length S, the draft model processes the full context and computes attention scores using a fixed number of lookahead tokens. As in prior work (Liu et al., 2025), lookahead is used to mitigate positional biases in attention distributions by aggregating signals from multiple decoding steps. Let the attention tensor be of shape *[N, L, S, H]*, where N is the number of lookahead tokens, L the number of transformer layers, S the prompt length, and H the number of attention heads. We compute token importance scores using the same max–mean aggregation strategy as speculative prefill: attention scores are first maximized over the layer and head dimensions, and then averaged across the lookahead dimension. This produces a single scalar importance score for each input token.

A.1.2 CHUNK-BASED PROMPT COMPRESSION

To reduce variance in token-level importance estimates, we perform prompt compression using a chunk-based selection strategy identical to speculative prefill. We first apply one-dimensional average pooling over the token importance scores. The input prompt is then partitioned into contiguous chunks, and the average importance score within each chunk is computed. Finally, the Top-K chunks are selected based on these average scores, yielding a compressed prompt.

A.2 BENCHMARK DETAILS

LongBench v1 and v2: We evaluate eight representative tasks spanning multi-document question answering (HotpotQA), query-based summarization (QMSum), passage retrieval (PassageRetrieval), long-form code understanding (LCC), and repository-level code reasoning (RepoBench) (Yang et al., 2018; Zhong et al., 2021; Guo et al., 2023; Liu et al., 2023). We use the full evaluation set for LongBench v2, which includes substantially longer and more diverse inputs than LongBench v1.

RULER and InfiniteBench: RULER focuses on extreme long-context reasoning tasks such as Needle-in-a-Haystack and long-context QA, while Code.Debug task from InfiniteBench requires identifying injected errors within large code repositories.

A.3 IMPLEMENTATION DETAILS

For all experiments, we use greedy decoding for answer generation. Discontinuous text spans produced by draft-model compression are separated using task-specific delimiter tokens. For LongBench and RULER results, we used `[...]` as the delimeter token. We vary the prompt keep rate across benchmarks while fixing the number of lookahead tokens to $N = 8$. For LongBench v1/v2 and RULER, we use a chunk size of 32 for top-K selection and apply one-dimensional average pooling with a kernel size of 13 to smooth attention scores prior to chunk selection.

For the code debugging task from InfiniteBench, we retain the above settings with two modifications. First, we use the delimiter token "// omitted" to explicitly mark non-contiguous code regions. Second, we increase the chunk size to 128 to better align with the structure of large code repositories. In addition, we make two evaluation-related adjustments to the InfiniteBench codebase: (i) we extend the answer extraction logic to include additional matching patterns to avoid false negatives, and (ii) we remove explicit `<think>` tags from DeepSeek-R1 responses prior to evaluation, which we find improves accuracy by preventing spurious mismatches during answer parsing. These adjustments apply equally to full prompt results also to make them directly comparable.

A.4 ADDITIONAL RESULT ANALYSIS

We report results on LongBench v1 and RULER in Tables 3 and 4. On LongBench v1, we apply cross-model speculative prefill to compress inputs to a fixed target length of 4k tokens for each target model, corresponding to an average keep rate of 45%. Across the selected LongBench v1 datasets, the average input length is 8805 tokens for LLaMA-8B and 8867 tokens for Qwen3-8B as target models. Draft models from both the Qwen and LLaMA families act as effective compressors, with cross-family prompt compression achieving accuracies close to the full-prompt baseline.

For RULER, we compress 128k-token inputs down to 16k tokens (12.5% keep rate). Notably, the full-prompt baseline underperforms speculative prefill in this setting. We attribute this to two main factors. First, speculative prefill performs implicit context denoising by retaining only salient regions of the input, which improves accuracy on retrieval-heavy, needle-in-a-haystack tasks. This denoising is missing for the reported baseline results. Second, making 128k full-prompt inference feasible with DeepSeek V3 requires combining it with SnapStream KV cache compression (Li et al., 2025) in our deployment setting. While necessary for memory reasons, KV cache compression can discard long-range information, leading to degradation in the full-prompt baseline, particularly for RULER which has retrieval heavy tasks.

In contrast, speculative prefill enables effective 128k input processing for DeepSeek V3, which is otherwise limited to a 32k context window on our hardware. By leveraging draft models (eg: LLaMA-8B) with native 128k context support, cross-family speculative prefill extends the usable

context length of the target model from 32k to 128k, enabling long-context inference that would otherwise be infeasible under deployment constraints. The performance with speculative prefill is also much better than the baseline results we get with SnapStream KV cache compression. These results highlight the practical benefit of speculative prefill in enabling long-context inference for high-capacity target models, while simultaneously improving performance through token-aware prompt compression.

A.5 Future research directions

While cross-family speculative prefill preserves near-baseline performance on most benchmarks, we observe a clear limitation on long-context code debugging. In Table 2, aggressive prompt compression (e.g., a 15% keep rate) leads to a noticeable accuracy drop for both DeepSeek-V3.1 and DeepSeek-R1. Specifically, DeepSeek-V3.1 decreases from 67.51% (full prompt) to 59.13 % at 15% keep rate, retaining $\sim$87.6% of baseline performance, and DeepSeek-R1 decreases from 74.37 % to 62.44 % at 15%, retaining $\sim$84.0% of baseline. These results fall outside the 90–100% baseline retention regime observed on many non-coding tasks.

We hypothesize that code debugging often depends on fine-grained syntactic and semantic dependencies that may be distributed across the repository; removing seemingly low saliency spans can break the local context required to pinpoint the injected error. Code debugging is less tolerant to extreme keep rates, and stronger structural constraints (e.g., dependency-aware span selection or minimum coverage of function-level context) may be required to maintain accuracy under heavy compression. We conclude that our current cross model speculative prefill framework is structure-dependent. These findings indicate that while cross-family speculative prefill is a general and effective prompt compression primitive, extending it with task-aware structural constraints is a promising direction for future work.

A.6 Qualitative Examples

Below we show a representative code debugging prompt after compression, using LLaMA-3.1-8B-Instruct as the draft model and DeepSeek-R1 as the target model. The keep rate is set to 30%. Delimiter tokens "// omitted" are highlighted in red to indicate breaks between non-contiguous text spans. The ground-truth answer is highlighted in blue, and the final question is placed at the end of the prompt. The full compressed prompt is approximately 33k tokens long; for clarity, we show only selected excerpts below.

There is ONLY ONE function in the large project that is deliberately made to include an obvious error. Please find the function that contains the most obvious errors. I will give you four options to narrow your scope. You can inspect the options and think. Eventually, tell me the answer using one single letter (A, B, C, or D).

Package: `pyarmor`

File: `pyarmor/cli/core/features.py`

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

# Each log
#    revision, age, (new features), (changed features), (removed features)
__CHANGE_LOGS__ = (
    (1, 0, (), (), ()),
)

class PyarmorFeature(object):

    def features(self):
        '''return features list from change logs'''
```

```
        result = set()
        [result.update(item[2]) for item in __CHANGE_LOGS__]
        return result


File: pyarmor/cli/core/__init__.py

#! /usr/bin/env python
# -*- coding: utf-8 -*-
#
__VERSION__ = '5.4.0'

def format_platform():
    import platform
    import sys
    from struct import calcsize

    def format_system():
        plat = platform.system().lower()
        plat = ('windows' if plat.startswith('cygwin') else
                'linux' if plat.startswith('linux') else
                'freebsd' if plat.startswith(
                    ('freebsd', 'openbsd', 'isilon onefs')) else plat)
        if plat == 'linux
// omitted

class PyarmorRuntime(object):

    @staticmethod
    def get(plat, extra=None, native=True):
        from os import scandir, path as os_path
        prefix = 'pyarmor_runtime'

        # Themida is only available for windows
        if extra == 'themida' and not plat.startswith('windows'):
            extra = None

        pkgpath = os_path.dirname(__file__)
        if native and not extra:
            path = pkgpath
            for entry in scandir(path):
                parts = entry.name.split('.')
                if parts[0] == prefix and parts[-1] in ('so', 'pyd', 'dylib'):
                    return entry.name, os_path.abspath(entry.path)

        dirnames = map_platform(plat).split('.')
        path = os_path.join(pkgpath, extra if extra else '', *dirnames)

// omitted
 getLogger('cli').info('fallback to pyarmor.cli.runtime==%s', ver)
            return PyarmorRuntime.get(plat, extra=extra)
        except ModuleNotFoundError:
            pass

def repack_carchive(executable, pkgfile, buildpath, obfpath, rtentry):
    pkgarch = CArchiveReader2(executable
// omitted
.dirname(executable), rtname)
            os.makedirs(dest, exist_ok=True)
            shutil.copy2(rtbinary, dest)

        repack_executable(executable, buildpath, obfpath, rtentry, codesign)

    def _fixup_darwin_rtbinary(self, rtbinary, pylib_name):
```

```
        '''Unused since Pyarmor 8.3.0'''
        from sys import version_info as pyver
        pylib = os.path.normpath(os.path.join('@rpath', pylib_name))
        output = check_output(['otool', '-L', rtbinary])
        for line in
// omitted
    main_entry()

@classmethod
def build_globfiles(cls, patterns, path=''):
    files = []
    n = len(path) + 1
    for x in patterns:
        for name in glob(os.path.join(path, x)):
            files.append(name[n:])
    return set(files)

def info(self):
    lines = []
    for k, v in Project.DEFAULT_VALUE:
        if k == 'build_time':
            v = time.asctime(time.gmtime(self[k]))
        else:
            v = str(self[k])
            n = 50
            if len(v) > n:
                v = v[:n] + '\ n %24s'%'' + v[n:]
        lines.append('%22s: %s' % (k, v))
    return '\ n'.join(lines)

if __name__ == '__main__':
    project = Project()
```

Which function has deliberate error?

- **A.** Resource.pkgname
- **B.** repack_carchive
- **C.** cmd_gen
- **D.** __init__

You should first find the functions in the options. Repeat their content, inspect through code, and at last give me your answer for the function that has the deliberate and obvious error in A, B, C, or D.