

WRITING IN THE MARGINS: BETTER INFERENCE PATTERN FOR LONG-CONTEXT RETRIEVAL

Anonymous authors

Paper under double-blind review

ABSTRACT

In this paper, we introduce Writing in the Margins (WiM), a new inference pattern for Large Language Models designed to optimize the handling of long input sequences in retrieval-oriented tasks. This approach leverages the chunked prefill of the key-value cache to perform segment-wise inference, which enables efficient processing of extensive contexts along with the generation and classification of intermediate information (“margins”) that guide the model towards specific tasks. This method increases computational overhead marginally while significantly enhancing the performance of off-the-shelf models without the need for fine-tuning. Specifically, we observe that WiM provides an average enhancement of 7.5% in accuracy for reasoning skills (HotpotQA, MultiHop-RAG) and a 30.0% increase in the F1-score for aggregation tasks (CWE). Additionally, we show how the proposed pattern fits into an interactive retrieval design that provides end-users with ongoing updates about the progress of context processing, and pinpoints the integration of relevant information into the final response. We release our implementation of WiM using Hugging Face Transformers library at [<anonymised URL>](#).

1 INTRODUCTION

The performance of Large Language Models (LLMs) tends to deteriorate when processing extensive inputs, a limitation linked directly to their fixed context window and attention mechanisms (Li et al., 2024; Liu et al., 2024). In particular, LLMs struggle with tasks involving long contexts, especially when the relevant information is embedded in larger volumes of text (Bai et al., 2024; Shaham et al., 2023). Recent research thus highlights the importance of improving model capabilities to handle more extensive datasets without losing accuracy or requiring exponential increases in computational resources.

There have been various attempts to extend the usable context window of LLMs, such as sparse attention (Tworkowski et al., 2023; Chen et al., 2024; Mohtashami & Jaggi, 2023), length extrapolation (Dai et al., 2019; Su et al., 2023; Peng et al., 2024), and context compression (Ge et al., 2024; Mu et al., 2023). Concurrently, the field has witnessed the rise of sophisticated prompting strategies like Chain of Thought (CoT) and related structured reasoning methods (Wei et al., 2022; Yao et al., 2023; Besta et al., 2024). These approaches have significantly enhanced LLMs’ ability to tackle complex tasks by systematically guiding the reasoning process through predefined structural patterns.

Our work bridges the gap between efficient transformers architecture research and development of new prompting strategies. Specifically, we identify a novel key-value (KV) cache aware reasoning pattern for existing off-the-shelf long context window LLMs in scenarios typical of retrieval-oriented tasks, where the context is substantial and the instructional prompt is comparatively short. We begin by recognizing that long-context prompts are commonly prefilled in the KV cache segment-wise in a process known as chunked prefill. From this insight, we introduce an inference pattern called Writing in the Margins (WiM), which concurrently generates query-based extractive summaries at each step of the prefill that are subsequently reintegrated at the end of the computation. We term these intermediate outputs “margins”, drawing inspiration from the practice of making margin notes for improved comprehension of long contexts in human reading. Using methodologies similar to “scratchpad” techniques, which meticulously record step-by-step calculations, we incorporate margin notes into the final segment predictions. We show that this technique, which adds only minimal

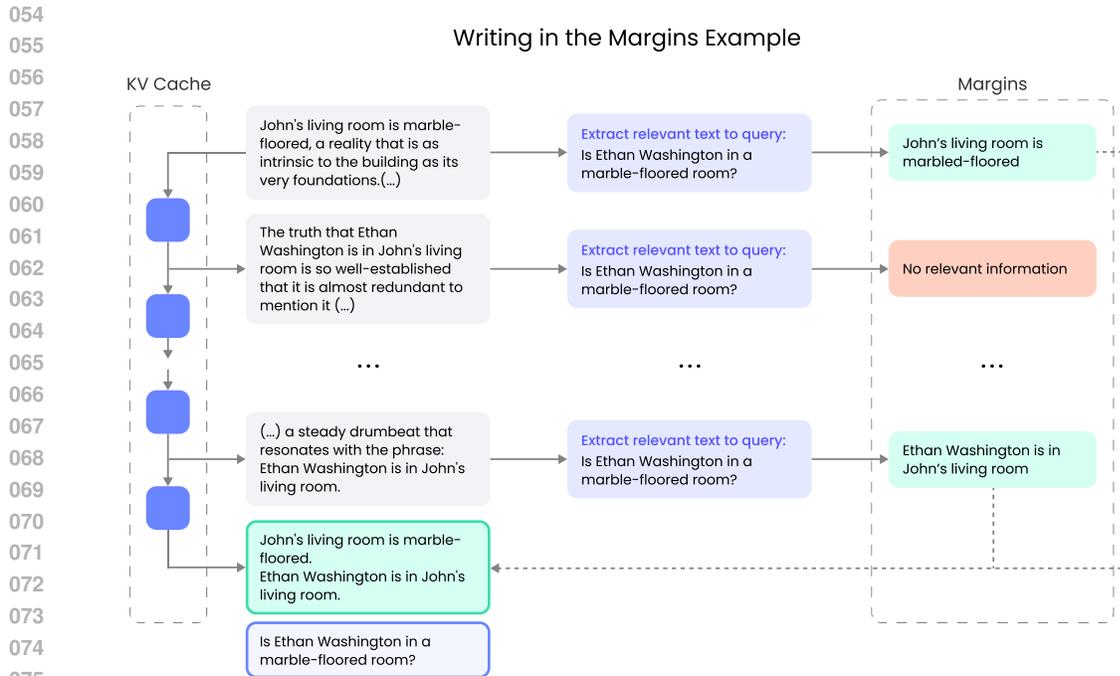


Figure 1: **Writing in the Margins inference pattern.** Prefilling KV cache by segments allows to both process the context segment by segment and generate intermediate extractive summaries which can improve the final prediction.

additional computation, significantly enhances long context comprehension. The WiM pattern can also provide end-users with real-time insights into computational progress through streamed margin notes, which ultimately help make AI decisions more transparent and explainable. This can enable users to (1) pinpoint the location of essential information and (2) reduce computational load by exiting early if the provided information satisfactorily addresses the query.

In Figure 1, we provide an illustrative example of WiM inference, which we encourage readers to reference as a practical demonstration to complement the formal algorithm description that will be presented in the following sections.

Our main contributions are as follows:

- We introduce a new inference pattern, Writing in the Margins (WiM), which achieves better performance on long-context window tasks with a relatively minor increase in computational cost.
- We demonstrate the application of WiM within an interactive long context retrieval setup, effectively increasing the transparency of the process and reducing the first response latency.
- We provide an implementation of this inference pattern using the Hugging Face Transformers library.

2 WRITING IN THE MARGINS

2.1 CHUNKED PREFILL

Typically, the process of inference for generative LLMs consists of two principal phases: the prefill phase and the decoding phase. When an LLM is requested to prefill a substantial prompt—in the range of hundreds of thousands of tokens—it is common practice to prefill the KV cache in chunks

Table 1: **Chunked Prefill.** Example of how the attention mask is set across different chunks during prefill iterations (first chunk on the left, second chunk on the right). Each new chunk needs to retain causality while attending to all previous chunks. Chunked prefill is mathematically equivalent to prefill without chunking.

	K0	K1	K2	K3		K0	K1	K2	K3	K4	K5	K6	K7
Q0	1	0	0	0	Q4	1	1	1	1	1	0	0	0
Q1	1	1	0	0	Q5	1	1	1	1	1	1	0	0
Q2	1	1	1	0	Q6	1	1	1	1	1	1	1	0
Q3	1	1	1	1	Q7	1	1	1	1	1	1	1	1

(Agrawal et al., 2024). This method is known as chunked prefill and is supported by many inference frameworks, including vLLM (vLLM, 2024).

Chunked prefill divides the prompt into fixed-size chunks to populate the KV cache at each layer of the Transformer model (Vaswani et al., 2017). The rationale for chunked prefill is to reduce overall memory usage, as the quadratic memory complexity of the attention mechanism during prefilling can be prohibitive for larger prompts. By splitting a prompt of length L into N chunks, each of size K , where $N = L/K$, the overall memory complexity of prefilling is reduced from $O(L^2)$ to $O(LK)$. The attention mask must be adjusted to allow each new chunk to attend to all tokens in the previous chunks while maintaining the causal structure only for the new chunk, as illustrated in Table 1.

Our work exploits the chunked prefill mechanism to generate intermediate “margins” that can then be appended to the prompt to better guide the model toward performing a specific task.

2.2 WRITING IN THE MARGINS

Consider a prompt P , composed of a context C , and an instruction I . Prefilling a decoder-only transformer model T directly with the entire prompt $T(P)$ is computationally inefficient when the prompt is long. Moreover, as shown in Liu et al. (2024), processing the entire prompt in one go can lead to mid-sequence forgetting.

To make this process more efficient, we implement the prefill technique described in the previous paragraph, where the context C is divided into N segments; i.e., $C = c_1 + c_2 + \dots + c_N$. For the first segment, the model T operates on chunk c_1 , resulting in output that includes past key values pkv_1 . The model continues onto the second segment with the pkv_1 cached, i.e., $T(\text{pkv}_1, c_2)$, effectively emulating the scenario of processing $T(c_1 + c_2)$ in one step. As the procedure progresses, each sequential chunk, c_k , is processed with prefilled past key values, noted as $T(\text{pkv}_{[1..k-1]}, c_k)$, mimicking an uninterrupted run of T on C .

The Writing in the Margins (WiM) strategy addresses potential mid-sequence forgetting issues by appending an extractive instruction I_A to each chunk, enhancing chunk-specific outputs. It transforms each step into $T(\text{pkv}_{[1..k-1]}, c_k + I_A)$, where the instruction I_A is embedded alongside each context chunk, then dropped from the KV cache before the next chunk prefilling. The instruction I_A is closely related to I - the model is asked to copy over all relevant to I information.

Intermediate outputs from each chunk are referred to as margin notes M_i , cumulatively forming N notes, described as $M = M_{[1..N]}$. Unhelpful notes, perhaps irrelevant to the instruction, are discarded, enhancing the final contextual construct to $C + M + I$, positioned advantageously towards the end to minimize mid-sequence memory loss. Intuitively, the model is allowed to use relevant intermediate predictions while answering the final query.

To summarize, we modify the chunked prefill algorithm by adding extra decoding steps (green in Table 2). Most of these steps can be efficiently batched with the original prefill steps. The query-relevant information extracted from these steps is then added at the end of the context but before the instruction (see Appendix A for a pseudocode example).

Table 2: **Batching Chunked Prefill Steps with WiM margin generation.** The inference for generative LLMs consists of two principal phases: the prefill phase (\dagger) and the decoding phase (\ddagger). The WiM algorithm adds extra decoding steps that mostly can be batched with chunked prefill steps. We keep margin notes M_i produced in extra steps (green) as plain text. We then prefill the model T with all relevant notes $M_{[1..N]}$ before the final instruction I .

step	Chunked Prefill	WiM	keep
1	$T(\emptyset, c_1)^\dagger$	$T(\emptyset, c_1)^\dagger$	pkv _[1]
2	$T(\text{pkv}_{[1]}, c_2)^\dagger$	$T(\text{pkv}_{[1]}, c_2)^\dagger$ $T(\text{pkv}_{[1]}, I_A)^{\dagger\ddagger}$	pkv _[1..2] M_1
\vdots	\vdots	\vdots	\vdots
N	$T(\text{pkv}_{[1..N-1]}, c_N)^\dagger$	$T(\text{pkv}_{[1..N-1]}, c_N)^\dagger$ $T(\text{pkv}_{[1..N-1]}, I_A)^{\dagger\ddagger}$	pkv _[1..N] M_{N-1}
N + 1		$T(\text{pkv}_{[1..N]}, I_A)^{\dagger\ddagger}$	M_N
N + 2	$T(\text{pkv}_{[1..N]}, I)^{\dagger\ddagger}$	$T(\text{pkv}_{[1..N]}, M_{[1..N]} + I)^{\dagger\ddagger}$	

Table 3: **Datasets** We curated four datasets to evaluate long context window LLMs. Each set consists of 100 examples, generated either using RULER code (\dagger) or by subsampling the longest examples from the original benchmark data (\clubsuit).

skill type	benchmark name	context length (tokens)	# examples
I	MultiHop-RAG \clubsuit (Tang & Yang, 2024)	13-32k	100
I	HotpotQA \dagger (Yang et al., 2018)	16k/ 32k /64k	100/ 100/ 100
II	SQuAD \dagger (Rajpurkar et al., 2018)	16k/ 32k/ 64k	100/ 100/ 100
III	CWE \dagger (Hsieh et al., 2024)	64k	100

3 EXPERIMENTAL SETUP

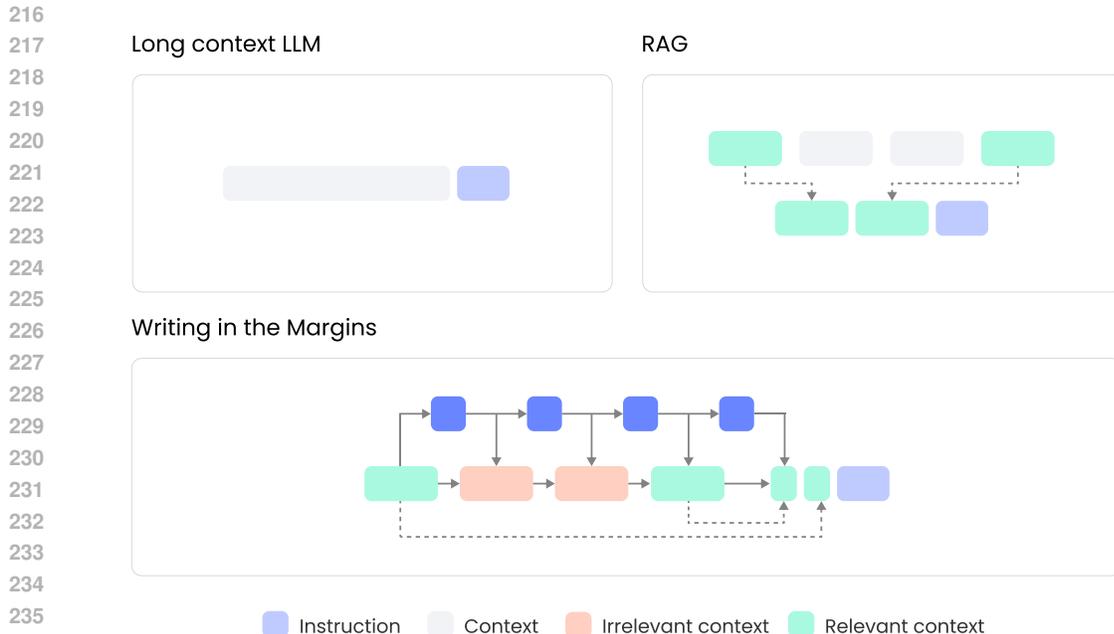
3.1 DATASETS

Following the RULER task categories (Hsieh et al., 2024), we measure the performance of an inference pattern on three types of skills: **(I) Multi-Hop Reasoning**, **(II) Needle Retrieval/ Single-Hop Reasoning**, and **(III) Aggregation**. Table 3 presents the curated long context datasets used to benchmark all LLMs:

In the following paragraph, we briefly introduce the benchmarks used in each category and describe our curating rationale.

- I. **Multi-Hop QA** The task aims to check the behavior of tracing entities with multi-hop connections based on the HotPotQA and MultiHop-RAG benchmarks (Tang & Yang, 2024; Yang et al., 2018). We used the RULER codebase¹ to generate a subset of 100 examples based on HotPotQA - a multi-hop queries sourced from Wikipedia articles. Following RULER, we simulated long context retrieval scenarios by generating examples in three length variants: 16k, 32k, 64k. We also selected the 100 longest examples in the range of 13k-33k tokens from MultiHop-RAG - a large collection of multi-hop queries based on English news articles.

¹<https://github.com/hsiehjackson/RULER>



238
239
240
241
242
243
244
245

Figure 2: **Design Comparison.** Design Comparison. Three inference designs for long contexts: (Top Left) Long Context LLM: Feeds entire context to the model without segmentation. (Top Right) Retrieval-Augmented Generation (RAG): Uses a retrieval method (e.g., cosine similarity) to select segments, which are then concatenated with task instructions for the model. (Bottom) Writing in the Margins (WiM): Divides and processes context by segments, prompting the model to generate auxiliary information from each, which is classified and potentially incorporated before the task description.

246
247
248
249
250
251
252

II. **Needle Retrieval/ Single-Hop Reasoning** In the context of a long context window, the Needle Retrieval and Single-Hop QA task can be jointly seen as a kind of filter benchmark, where the task is to filter irrelevant content and either copy or transform the relevant information. We used the RULER code to generate examples based on SQuAD (Rajpurkar et al., 2018) in three context length variants: $16k$, $32k$ and $64k$, collecting 100 datapoints in each variant.

253
254
255
256
257
258
259

III. **Aggregation** This task evaluates a model’s ability to aggregate relevant information across a long-range context, using the Common Words Extraction (CWE) benchmark (Hsieh et al., 2024). In this benchmark, word distribution numbers are fixed with the sequence length, using 100 examples averaging $64k$ tokens each. Common words appear 500 times, while uncommon words appear no more than 50 times. Task instructions were adapted to include word occurrence counts to facilitate segment aggregation.

260 261 3.2 LONG CONTEXT WINDOW LLMs

262
263
264
265
266
267

We selected seven off-the-shelf models that officially support context windows up to $128k$ tokens: Phi-3-small-128k-instruct (Abdin et al., 2024), Qwen2-7B-Instruct (Yang et al., 2024), Meta-Llama-3.1-8B-Instruct (Dubey et al., 2024), Phi-3-medium-128k-Instruct (Abdin et al., 2024), Palmyra-4-Chat-128K (Writer’s proprietary model), Meta-Llama-3.1-70B-Instruct (Dubey et al., 2024), Qwen2-72B-Instruct (Yang et al., 2024).

268
269

In all experiments, we used half precision models with identical sampling parameters — specifically, a temperature setting of 0.0 and $2k$ maximum new tokens. We used 0-shot prompts for all benchmarks. In MultiHop-RAG, HotPotQA and SQuAD experiments, we applied the same model-

independent preprocessing step: we used nltk (Bird et al., 2009) to split the context into sentences, then grouped them in segments no longer than 4096 tokens. This resulted in 4 – 16 margin notes per datapoint. In CWE, where the datapoints contain only numbered words, we exchanged nltk for naive words split by space and used 8192 segment length, which gave on average 8 margins per sample. We chose to count tokens using GPT-4 tiktoken tokenizer² since this choice does not favour any of the evaluated models’ tokenizers.

In each case, we measured the relative differences of WiM pattern scores with respect to the following two baselines:

- **Long Context LLM (LLM)** - all context without segmentation is fed to the LLM.
- **Retrieval Augmented Generation (RAG)** - segments are selected based on a retriever (ex. cosine similarity between vector representations of the query and the segment), then all selected segments and the task instruction are concatenated and fed to an LLM.

In order to make the results more comparable, we replaced the retriever in RAG with the classifier used in WiM. We expect the RAG results to be lower in the real RAG systems (especially for longer segment lengths), as vectorization is a form of lossy compression. All three inference patterns, including WiM, are presented in Figure 2.

3.3 EVALUATION

3.3.1 PREDICTION

In the margin accumulation step, in order to distinguish the content of the margins from the original context, and to maintain the document’s logic and structure, we explicitly named the writing-in-the-margins strategy by reformatting the margins the following way:

```
I asked my assistant to read and analyse the above content page by page
  ↪ to help you complete this task. Those are margin notes left on each
  ↪ page:
  ```text
Page 0:
QUERY: {query}
ANSWER: {M_i}
Page 1:
QUERY: {query}
ANSWER: {M_j}
...

```

The output is appended at the end of the final prompt. Full prompts are shown in Appendix C.

#### 3.3.2 SCORING

We used the same 3-shot prompt with GPT-4-turbo (OpenAI, 2023) and greedy sampling to evaluate models’ accuracy in HotpotQA, MultiHop-RAG and SQuAD benchmarks. For the CWE benchmark we adjusted the prompt and examples to calculate precision (P), recall (R) and F1-score. Both prompts are shown in Appendix C.

## 4 RESULTS

### 4.1 MULTI-HOP REASONING

Detailed results for all experiments are presented in Table 4. Notably, for almost all evaluated models, WiM improves multi-hop reasoning abilities, on average giving a 7.5% boost with respect to the Long Context LLM inference and 9% with respect to RAG. The most significant performance boost is observed in smaller models — replacing a vanilla Phi-3-small-128k-instruct inference with WiM leads to 19% improvement in MultiHop-RAG benchmark and 12% in HotpotQA.

<sup>2</sup><https://github.com/openai/tiktoken>

Table 4: **Main Results** We show results for seven models and four benchmarks, using accuracy for all but CWE, where precision, recall, and F1-score were used. Aggregated results indicate WiM excels in multi-hop reasoning and summarization tasks (HotpotQA, Multihop-RAG, CWE), while performance in single-hop reasoning (SQuAD) varies by model.

Model	Pattern	HotpotQA			MultiHop RAG	SQuAD			CWE			Average
		Context: 16k	32k	64k	13-32k	16k	32k	64k	64k			Excl. CWE
		Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	P	R	F1	Acc.
Phi-3-small-128k-instruct	LLM	0.47	0.55	0.48	0.58	<b>0.81</b>	0.75	<b>0.79</b>	<b>0.77</b>	<b>0.77</b>	<b>0.77</b>	0.52
	RAG	0.55	0.56	0.50	0.70	<b>0.81</b>	<b>0.78</b>	<b>0.79</b>	0.65	0.64	0.65	<b>0.58</b>
	WiM	<b>0.66</b>	<b>0.64</b>	<b>0.56</b>	<b>0.77</b>	0.65	0.74	0.64	0.70	0.69	0.69	<b>0.66</b>
Qwen2-7B-Instruct	LLM	0.62	0.59	0.39	0.83	0.81	0.71	0.57	0.46	0.46	0.46	0.61
	RAG	0.54	0.55	<b>0.56</b>	0.77	<b>0.87</b>	<b>0.84</b>	<b>0.86</b>	0.49	0.49	0.49	0.61
	WiM	<b>0.69</b>	<b>0.66</b>	<b>0.56</b>	<b>0.92</b>	0.83	0.80	0.74	<b>0.69</b>	<b>0.67</b>	<b>0.68</b>	<b>0.71</b>
Meta-Llama-3.1-8B-Instruct	LLM	0.65	0.64	0.60	0.85	<b>0.90</b>	<b>0.92</b>	0.87	0.22	0.21	0.22	0.69
	RAG	0.67	0.65	0.59	0.77	0.87	0.91	<b>0.91</b>	0.47	0.47	0.47	0.67
	WiM	<b>0.77</b>	<b>0.71</b>	<b>0.73</b>	<b>0.86</b>	0.88	0.85	0.82	<b>0.94</b>	<b>0.93</b>	<b>0.93</b>	<b>0.77</b>
Phi-3-medium-128k-instruct	LLM	0.57	0.53	0.48	0.80	0.84	0.72	0.70	<b>0.91</b>	<b>0.91</b>	<b>0.91</b>	0.60
	RAG	0.50	0.55	0.51	0.78	<b>0.86</b>	<b>0.82</b>	<b>0.83</b>	<b>0.91</b>	<b>0.91</b>	<b>0.91</b>	0.59
	WiM	<b>0.63</b>	<b>0.67</b>	<b>0.57</b>	<b>0.93</b>	0.81	0.80	0.77	0.90	0.90	0.90	<b>0.70</b>
Palmyra-4-Chat-128K	LLM	<b>0.70</b>	0.60	0.57	0.85	<b>0.84</b>	0.76	0.73	0.76	0.77	0.76	0.68
	RAG	0.59	0.54	0.55	0.78	0.74	0.70	0.69	<b>0.80</b>	<b>0.80</b>	<b>0.80</b>	0.62
	WiM	0.69	<b>0.63</b>	<b>0.66</b>	<b>0.86</b>	0.78	<b>0.77</b>	<b>0.74</b>	0.77	0.77	0.77	<b>0.71</b>
Meta-Llama-3.1-70B-Instruct	LLM	<b>0.80</b>	0.74	0.70	<b>0.91</b>	<b>0.93</b>	0.85	0.87	0.37	0.36	0.36	<b>0.79</b>
	RAG	0.73	0.72	0.63	0.80	0.90	<b>0.92</b>	<b>0.95</b>	0.66	0.65	0.66	0.72
	WiM	0.79	<b>0.76</b>	<b>0.71</b>	0.89	0.90	0.90	0.82	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	0.79
Qwen2-72B-Instruct	LLM	0.75	0.72	0.57	<b>0.88</b>	0.91	0.78	0.76	0.42	0.36	0.39	0.73
	RAG	0.70	0.66	<b>0.70</b>	0.80	<b>0.92</b>	0.87	<b>0.91</b>	0.75	0.75	0.75	0.72
	WiM	<b>0.80</b>	<b>0.79</b>	<b>0.70</b>	<b>0.88</b>	0.88	<b>0.88</b>	0.87	<b>0.98</b>	<b>0.98</b>	<b>0.98</b>	<b>0.79</b>
Average	LLM	0.65	0.62	0.54	0.81	<b>0.86</b>	0.78	0.76	0.56	0.55	0.55	0.66
	RAG	0.61	0.60	0.58	0.77	0.85	<b>0.83</b>	<b>0.85</b>	0.68	0.67	0.68	0.64
	WiM	<b>0.72</b>	<b>0.69</b>	<b>0.64</b>	<b>0.87</b>	0.82	0.82	0.77	<b>0.85</b>	<b>0.85</b>	<b>0.85</b>	<b>0.73</b>

By looking at different length variants of HotpotQA (16k, 32k, 64k) we see that all patterns lose accuracy as we add more context (LLM: from 0.65 to 0.54, RAG: from 0.61 to 0.58, WiM: from 0.72 to 0.64). This observation aligns with the notion that extending the context length in models degrades the performance of complex reasoning tasks. However, using WiM allows us to maintain almost the same accuracy for 64k as the LLM achieves on 16k.

#### 4.2 NEEDLE RETRIEVAL AND SINGLE-HOP QUESTION ANSWERING

Analysis of the SQuAD benchmark results shows that all scores are distributed across similar values with a slight preference for RAG. WiM prompting increase verbosity of LLMs, which is distracting for SQuAD expecting short answers. Nevertheless, we see that replacing an LLM with the WiM pattern consistently improves accuracy in SQuAD by 2% – 17% for Qwen2-7B-Instruct, whereas LLM is a preferred inference pattern for 16k context window for 4 out of 7 tested models.

Unsurprisingly, RAG emerges as the most optimal pattern for six out of seven evaluated models when extending the context length to 64k tokens in SQuAD. Indeed, for single-hop reasoning tasks, if the filtering process is successful (here we approximate the retriever by an LLM classifier), the challenge is reduced to a trivial task of retrieving a needle from a context window of 4096 tokens. However, this assumption in the RAG setup is overly optimistic because the LLMs used in our experiment are at least 7B in model parameters, and such large models are not typically used as retrievers. In practical scenarios, one might expect the results to be even more favorable for both LLM and WiM compared to RAG.

#### 4.3 AGGREGATION

The pattern across the data indicates that WiM either matches or substantially boosts the aggregation skills of off-the-shelf models, giving an LLM on average a 30% increase in F1-score for the CWE benchmark, and outperforming RAG by 17%.

Table 5: **Ablation: Filtering Margins and Content Compression** Removing irrelevant margins improves results for most models across HotpotQA, Multihop-RAG, and SQuAD benchmarks. Using both margins and full context generally boosts scores, despite performance drops in models with longer inputs. Results aggregated over HotpotQA, Multihop-RAG, and SQuAD benchmarks.

Model	Margin Filter		Content Compression		
	Unfiltered	Filtered (WiM)	Only Margins	Only Context	Both (WiM)
Phi-3-small-128k-instruct	0.54	<b>0.58</b>	<b>0.60</b>	0.55	0.58
Qwen2-7B-Instruct	0.63	<b>0.65</b>	0.62	0.56	<b>0.65</b>
Meta-Llama-3.1-8B-Instruct	<b>0.70</b>	<b>0.70</b>	0.68	0.68	<b>0.70</b>
Phi-3-medium-128k-instruct	0.64	<b>0.65</b>	0.57	0.58	<b>0.65</b>
Palmyra-4-Chat-128K	0.55	<b>0.64</b>	0.53	0.63	<b>0.64</b>
Meta-Llama-3.1-70B-Instruct	<b>0.73</b>	0.72	<b>0.72</b>	<b>0.72</b>	<b>0.72</b>
Qwen2-72B-Instruct	0.71	<b>0.72</b>	<b>0.72</b>	0.67	<b>0.72</b>

We observe that CWE results can be grouped into four classes, which surprisingly tend to align more with the model families than with the model sizes. Models like Meta-Llama-3.1-8B-Instruct and Meta-Llama-3.1-70B-Instruct achieve a remarkably significant boost in F1-score when using WiM across all context lengths, reaching up to 72% compared to the LLM baseline. Conversely, models like Phi-3-small-128k-instruct and Phi-3-medium-128k-instruct consistently prefer the vanilla LLM inference. Meanwhile, Qwen2-7B-Instruct and Qwen2-72B-Instruct point to WiM as the most optimal pattern, showing a moderate improvement ranging from 22% to 59%. RAG is preferred only by Palmyra-4-Chat-128K, which outperforms the rest by 3% – 4%.

After reviewing all comments provided by GPT-4-turbo during the evaluation of each data point, we observe that models often resort to writing Python code to solve the problem (18.5% of all answers), leading to incorrect or generic answers (resulting in on average 20% drop in F1-score).

## 5 ABLATION STUDY

### 5.1 NO MARGINS FILTERING

In this experiment, we excluded the margins classifier from the WiM pipeline, which resulted in all extractive summaries being appended directly to the context.

Table 5 presents the accuracy scores aggregated over three benchmarks: HotpotQA, MultiHop-RAG, SQuAD. Including all margins in the context decreases the accuracy by up to 8% compared to the original WiM pipeline. This effect is analogous to negative instruction manipulation, akin to telling the model to “forget all previous instructions”. Ultimately, filtering margins—especially when combined with the margin generation step—not only saves computation by allowing irrelevant margins to be dropped, but also improves the overall performance.

### 5.2 REPLACING THE CONTENT BY MARGINS

An effective approach to reduce computational demands is to eliminate the KV cache in the final step, relying only on extracted positive margins. This method compresses the long context document based on the query. Although retaining the full context may capture answers better, increased input length has been shown to reduce model performance.

Table 5 also presents aggregated results for HotpotQA, MultiHop-RAG and SQuAD, demonstrating that incorporating both margins and the complete document consistently maximized the performance for almost all evaluated models, except for Meta-Phi-3-small-128k-instruct. Employing a query-based extractive summary—specifically, using only the content from margins—gave mixed results across all models; e.g., Meta-Llama-3.1-70B-Instruct scores were consistent across all metrics (0.72), while Palmyra-4-Chat-128K scores saw a decrease from 0.64 to 0.53. On the other hand, the model Phi-3-small-128k-instruct experienced an increase from 0.58 to 0.6. We hypothesize that these outcomes might vary depending on the specific task at hand. It is plausible that for tasks such

432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485

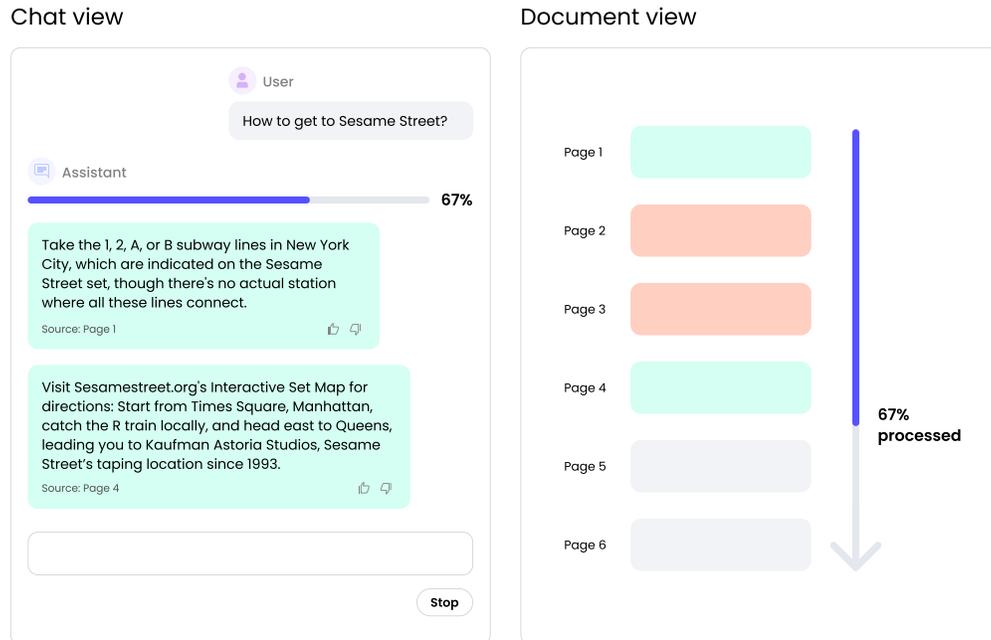


Figure 3: **WiM interactive retrieval design.** The right side displays the document view, showing processed segments, which can be labeled for relevance by the LLM classifier. The left side features a chat view with a progress bar for segment processing. Users can interact by approving or rejecting margins, and these interactions influence the final response. Each margin corresponds to a specific document segment.

as filtering and improving recall (i.e., when models are fine-tuned for margin generation and classification tasks), using margins could prove beneficial as the filtered-out content would be entirely irrelevant.

## 6 INTERACTIVE RETRIEVAL

**Explainability** The design principles behind WiM focus not just on enhancing final benchmark performance, but also on improving the user experience. By presenting intermediate computation steps, WiM renders the decision-making process of LLMs transparent. This clarity in the model’s reasoning process aids not only in debugging but also provides insights that are crucial for both end-users and developers, ensuring outputs that are both understandable and reliable.

**Latency** Handling long documents can degrade user experience due to significant latency, as the model becomes unresponsive during processing, which can take minutes without clear indications of wait time. Our design addresses this by providing relevant information during processing and by segment-wise processing that incorporates a progress bar, thus reducing the initial response latency.

**Early exit** WiM also offers an “early exit” option, allowing users to stop the computation if they find a satisfactory answer within any of the displayed margins. For example, in single-hop question-answering scenarios, once the answer is found in a particular section, there is no need to process further.

**Human in the Loop** Users have the ability to improve the decision-making process by adding labels to the margins displayed in WiM. In this design, the final answer considers both the full context and the user-labeled margins. Users can evaluate and label the streamed margins (e.g., with a thumbs up or down), and these inputs could be reintegrated into the final decision-making step. The proposed design, including this feedback loop, is illustrated in Figure 3.

## 7 RELATED WORK

**External Memory and Retrieval Methods** Memory augmentation in Large Language Models (LLMs) involves integrating external memory banks, such as k-nearest neighbor (k-NN) models, to use textual similarities for generating context-aware completions (Khandelwal et al., 2020). These k-NN based LLMs excel in managing irregular patterns and factual data (Daelemans et al., 1999). Additionally, approaches like Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) and “Entities as Experts” (Férvy et al., 2020) link LLMs with external data sources—ranging from structured knowledge graphs (Liu et al., 2022) to learned entity embeddings. Such methods allow LLMs to access and utilize external information to enhance response accuracy and relevance.

**Scratchpad Mechanisms** A method for intermediate computation in LLMs involves the use of “scratchpads” or CoT (Wei et al., 2022) as a method for improving handling of sustained multi-step computations. Adopted from the findings of “Show Your Work: Scratchpads for Intermediate Computation with Language Models” (Nye et al., 2022) this method enables LLMs to show their logic step-by-step, similar to a human using paper to jot down interim calculations. By training Transformers to sequentially output the results of intermediate steps rather than only final answers, LLMs demonstrate enhanced performance on complex tasks that go beyond single-step reasoning, such as long addition and program execution. This method not only helps the model maintain and extend context dynamically but also aids in debugging and understanding model decisions (Austin et al., 2021). Further studies into length generalization have demonstrated that traditional fine-tuning techniques on tasks requiring such generalizations often encounter significant limitations (Anil et al., 2022). By integrating scratchpad-like methodologies, these language models can achieve a notable improvement in handling progressively longer text spans. This enhancement proves particularly valuable for challenges such as theorem proving and extensive text synthesis. Here, the in-context learning combined with the sequential output of computed steps substantially bolsters task accuracy and model robustness (Chen et al., 2021; Wu et al., 2021).

**Context Aggregation** Efficiency in context aggregation for LLMs have evolved with methods like Fusion-in-Decoder (FiD) and Map Reduce. FiD, used in models such as T5 and BART, consolidates contextual embeddings via encoder and decoder components to ensure comprehensive information integration (Ivgy et al., 2023; Izacard & Grave, 2021). Conversely, LangChain’s Map Reduce processes segments in parallel to quickly synthesize responses into a refined final output (Chase, 2022). Parallel Context Windows (PCW) and Naive Bayes Context Extension (NBCE) further enhance handling of extended contexts by partitioning these into smaller segments for efficient parallel processing, optimizing both processing speed and response relevance (Su et al., 2024; Ratner et al., 2023).

## 8 CONCLUSION

In this paper, we have introduced a new inference pattern called Writing in the Margins (WiM), which leverages chunked prefill to add only a marginal computational cost, emulating the human behavior of making notes in the margins. We demonstrated that this inference pattern significantly boosts the performance of off-the-shelf models across various long-context, retrieval-oriented tasks, including multi-hop reasoning (by 7.5% in HotpotQA, MultiHop-RAG), and aggregation (by 30.0% in CWE). Remarkably, this method does not require finetuning and is compatible with any transformer model.

Additionally, our approach enhances end-user experience by making context processing more transparent. By streaming “margins” that influence final predictions, our design supports early engagement. WiM differs from traditional long-context methods by allowing immediate streaming of relevant margins after segment processing, improving latency and reducing computational demands through an early exit strategy. This feature facilitates human-in-the-loop involvement in LLM decision-making, boosting interaction and intervention opportunities.

Our innovation decouples training and inference, building on ideas like CoT. By merging KV cache management with targeted prompting strategies, our approach complements existing prompt-based methods, aiming to initiate research into KV cache-aware prompting. This could improve LLM reasoning abilities and add a layer of interpretability.

**Reproducibility Statement** We have made the code for reproducing our results available through the HuggingFace transformer library at an <anonymised URL>. The evaluation data can be obtained from the HuggingFace Hub under MultiHop-RAG<sup>3</sup>, or it can be generated using the RULER code<sup>4</sup> for datasets such as SQuAD, HotpotQA, and CWE, with the specific parameters detailed in our paper. Both sources are provided under a permissive license. Furthermore, we have also disclosed the inference parameters and prompts needed to replicate our results.

## REFERENCES

- Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Qin Cai, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Yen-Chun Chen, Yi-Ling Chen, Parul Chopra, Xiyang Dai, Allie Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Victor Fragoso, Dan Iter, Mei Gao, Min Gao, Jianfeng Gao, Amit Garg, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Ce Liu, Mengchen Liu, Weishung Liu, Eric Lin, Zeqi Lin, Chong Luo, Piyush Madan, Matt Mazzola, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacrose, Shital Shah, Ning Shang, Hiteshi Sharma, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Xin Wang, Lijuan Wang, Chunyu Wang, Yu Wang, Rachel Ward, Guanhua Wang, Philipp Witte, Haiping Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Sonali Yadav, Fan Yang, Jianwei Yang, Ziyi Yang, Yifan Yang, Donghan Yu, Lu Yuan, Chengruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL <https://arxiv.org/abs/2404.14219>.
- Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, Santa Clara, CA, 2024. USENIX Association. URL <https://www.usenix.org/conference/osdi24/presentation/agrawal>.
- Cem Anil, Yuhuai Wu, Anders Johan Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Venkatesh Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL <https://openreview.net/forum?id=zSkYVeX7bC4>.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL <https://arxiv.org/abs/2108.07732>.
- 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)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL <https://aclanthology.org/2024.acl-long.172>.
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Giniński, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, March 2024.

<sup>3</sup><https://huggingface.co/datasets/yixuantt/MultiHopRAG>

<sup>4</sup><https://github.com/hsiehjackson/RULER>

- 594 ISSN 2159-5399. doi: 10.1609/aaai.v38i16.29720. URL [http://dx.doi.org/10.1609/](http://dx.doi.org/10.1609/aaai.v38i16.29720)  
595 [aaai.v38i16.29720](http://dx.doi.org/10.1609/aaai.v38i16.29720).  
596
- 597 Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing*  
598 *text with the natural language toolkit*. ” O’Reilly Media, Inc.”, 2009.
- 599 Harrison Chase. Langchain. <https://github.com/langchain-ai/langchain>, 10  
600 2022.  
601
- 602 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared  
603 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri,  
604 Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan,  
605 Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian,  
606 Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fo-  
607 tios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex  
608 Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders,  
609 Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec  
610 Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob Mc-  
611 Grew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large  
612 language models trained on code, 2021.
- 613 Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Lon-  
614 gloRA: Efficient fine-tuning of long-context large language models. In *The Twelfth International*  
615 *Conference on Learning Representations*, 2024. URL [https://openreview.net/forum?](https://openreview.net/forum?id=6PmJoRfdaK)  
616 [id=6PmJoRfdaK](https://openreview.net/forum?id=6PmJoRfdaK).
- 617 W. Daelemans, A. van den Bosch, and J. Zavrel. Forgetting exceptions is harmful in language  
618 learning. *Machine Learning*, 34(1-3):11–41, 1999. ISSN 0885-6125. Pagination: 32.
- 619 Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov.  
620 Transformer-XL: Attentive language models beyond a fixed-length context. In Anna Korhonen,  
621 David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Associ-*  
622 *ation for Computational Linguistics*, pp. 2978–2988, Florence, Italy, July 2019. Association for  
623 Computational Linguistics. doi: 10.18653/v1/P19-1285. URL [https://aclanthology.](https://aclanthology.org/P19-1285)  
624 [org/P19-1285](https://aclanthology.org/P19-1285).
- 625
- 626 Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha  
627 Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony  
628 Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark,  
629 Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere,  
630 Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris  
631 Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong,  
632 Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny  
633 Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino,  
634 Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael  
635 Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Ander-  
636 son, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah  
637 Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan  
638 Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Ma-  
639 hadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy  
640 Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak,  
641 Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Al-  
642 wala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini,  
643 Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yearly, Laurens van der  
644 Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo,  
645 Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Man-  
646 nat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova,  
647 Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal,  
Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur  
Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhar-  
gava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong,

648 Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic,  
649 Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sum-  
650 baly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa,  
651 Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang,  
652 Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende,  
653 Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney  
654 Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom,  
655 Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta,  
656 Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petro-  
657 vic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang,  
658 Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuwei Wang, Yaelle Goldschlag, Yashesh Gaur,  
659 Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre  
660 Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha  
661 Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay  
662 Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda  
663 Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew  
664 Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita  
665 Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh  
666 Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De  
667 Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon  
668 Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina  
669 Mejia, Changan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris  
670 Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel  
671 Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Di-  
672 ana Liskovich, Didem Foss, Dingkan Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa  
673 Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Es-  
674 teban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel,  
675 Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Flo-  
676 rez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Her-  
677 man, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou,  
678 Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren,  
679 Hunter Goldman, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman,  
680 James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer  
681 Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe  
682 Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie  
683 Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun  
684 Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal  
685 Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva,  
686 Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian  
687 Khabza, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson,  
688 Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Ke-  
689 neally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel  
690 Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mo-  
691 hammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navy-  
692 ata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong,  
693 Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli,  
694 Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux,  
695 Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao,  
696 Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li,  
697 Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott,  
698 Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Sa-  
699 tadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lind-  
700 say, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang  
701 Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen  
Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho,  
Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser,  
Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Tim-  
othy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan,  
Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihalescu,

- 702 Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xi-  
703 aocheng Tang, Xiaofang Wang, Xiaojuan Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao,  
704 Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin  
705 Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuze He, Zach Rait, Zachary DeVito, Zef Rosnbrick,  
706 Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- 707  
708 Thibault Févry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski.  
709 Entities as experts: Sparse memory access with entity supervision. In Bonnie Webber, Trevor  
710 Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods*  
711 *in Natural Language Processing (EMNLP)*, pp. 4937–4951, Online, November 2020. Association  
712 for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.400. URL <https://aclanthology.org/2020.emnlp-main.400>.
- 713  
714 Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for  
715 context compression in a large language model, 2024. URL [https://arxiv.org/abs/](https://arxiv.org/abs/2307.06945)  
716 [2307.06945](https://arxiv.org/abs/2307.06945).
- 717  
718 Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and  
719 Boris Ginsburg. RULER: What’s the real context size of your long-context language models? In  
720 *First Conference on Language Modeling*, 2024. URL [https://openreview.net/forum?](https://openreview.net/forum?id=kIoBbc76Sy)  
721 [id=kIoBbc76Sy](https://openreview.net/forum?id=kIoBbc76Sy).
- 722  
723 Maor Ivgi, Uri Shaham, and Jonathan Berant. Efficient long-text understanding with short-text  
724 models. *Transactions of the Association for Computational Linguistics*, 11:284–299, 2023.
- 725  
726 Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open  
727 domain question answering. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (eds.), *Proceed-*  
728 *ings of the 16th Conference of the European Chapter of the Association for Computational Lin-*  
729 *guistics: Main Volume*, pp. 874–880, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.74. URL [https://aclanthology.org/2021.](https://aclanthology.org/2021.eacl-main.74)  
730 [eacl-main.74](https://aclanthology.org/2021.eacl-main.74).
- 731  
732 Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization  
733 through memorization: Nearest neighbor language models. In *International Conference on Learn-*  
734 *ing Representations*, 2020. URL <https://openreview.net/forum?id=Hk1BjCEKvH>.
- 735  
736 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E.  
737 Gonzalez, Haoteng Zhang, and Ion Stoica. Efficient memory management for large language  
738 model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems*  
739 *Principles*, SOSP ’23, New York, NY, USA, 2023. Association for Computing Machinery. doi:  
740 [10.1145/3600006.3613165](https://doi.org/10.1145/3600006.3613165). URL <https://doi.org/10.1145/3600006.3613165>.
- 741  
742 Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman  
743 Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel,  
744 and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In  
745 H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural*  
746 *Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc.,  
747 2020. URL [https://proceedings.neurips.cc/paper\\_files/paper/2020/](https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf)  
748 [file/6b493230205f780e1bc26945df7481e5-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf).
- 749  
750 Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. LooGLE: Can long-context lan-  
751 guage models understand long contexts? In Lun-Wei Ku, Andre Martins, and Vivek Sriku-  
752 mar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Lin-*  
753 *guistics (Volume 1: Long Papers)*, pp. 16304–16333, Bangkok, Thailand, August 2024. As-  
754 sociation for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.859. URL <https://aclanthology.org/2024.acl-long.859>.
- 755  
756 Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and  
757 Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the*  
758 *Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl.a.00638. URL  
759 <https://aclanthology.org/2024.tacl-1.9>.

- 756 Qi Liu, Dani Yogatama, and Phil Blunsom. Relational memory-augmented language mod-  
757 els. *Transactions of the Association for Computational Linguistics*, 10:555–572, 2022. doi:  
758 10.1162/tacl.a.00476. URL <https://aclanthology.org/2022.tacl-1.32>.  
759
- 760 Amirkeivan Mohtashami and Martin Jaggi. Random-access infinite context length for transformers.  
761 In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=7eHn64wOVy>.  
762
- 763 Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens.  
764 In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=2DtXPCL3T5>.  
765  
766
- 767 Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David  
768 Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Au-  
769 gustus Odena. Show your work: Scratchpads for intermediate computation with language models.  
770 In *Deep Learning for Code Workshop*, 2022. URL <https://openreview.net/forum?id=HBlx2idbkbq>.  
771
- 772 OpenAI. Gpt 4., 2023. URL <https://openai.com/research/gpt-4>.  
773
- 774 Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context win-  
775 dows extension of large language models. In *The Twelfth International Conference on Learning*  
776 *Representations*, 2024. URL <https://openreview.net/forum?id=wHBfxhZulu>.  
777
- 778 Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don’t know: Unanswerable ques-  
779 tions for SQuAD. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th An-  
780 nual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp.  
781 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi:  
782 10.18653/v1/P18-2124. URL <https://aclanthology.org/P18-2124>.
- 783 Nir Ratner, Yoav Levine, Yonatan Belinkov, Ori Ram, Inbal Magar, Omri Abend, Ehud Karpas,  
784 Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. Parallel context windows for large  
785 language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceed-  
786 ings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1:  
787 Long Papers)*, pp. 6383–6402, Toronto, Canada, July 2023. Association for Computational Lin-  
788 guistics. doi: 10.18653/v1/2023.acl-long.352. URL <https://aclanthology.org/2023.acl-long.352>.  
789
- 790 Uri Shaham, Maor Ivgi, Avia Efrat, Jonathan Berant, and Omer Levy. ZeroSCROLLS: A zero-shot  
791 benchmark for long text understanding. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.),  
792 *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 7977–7989, Singa-  
793 pore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.  
794 findings-emnlp.536. URL <https://aclanthology.org/2023.findings-emnlp.536>.  
795  
796
- 797 Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Reformer: En-  
798 hanced transformer with rotary position embedding, 2023. URL <https://arxiv.org/abs/2104.09864>.  
799
- 800 Jianlin Su, Murtadha Ahmed, Bo Wen, Luo Ao, Mingren Zhu, and Yunfeng Liu. Naive Bayes-  
801 based context extension for large language models. In Kevin Duh, Helena Gomez, and Steven  
802 Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the As-  
803 sociation for Computational Linguistics: Human Language Technologies (Volume 1: Long Pa-  
804 pers)*, pp. 7791–7807, Mexico City, Mexico, June 2024. Association for Computational Linguis-  
805 tics. doi: 10.18653/v1/2024.naacl-long.431. URL <https://aclanthology.org/2024.naacl-long.431>.  
806  
807
- 808 Yixuan Tang and Yi Yang. Multihop-RAG: Benchmarking retrieval-augmented generation for multi-  
809 hop queries. In *First Conference on Language Modeling*, 2024. URL <https://openreview.net/forum?id=t4eB3zYWBK>.

- 810 Szymon Tworowski, Konrad Staniszewski, Mikołaj Patek, Yuhuai Wu, Henryk Michalewski, and  
811 Piotr Miłoś. Focused transformer: Contrastive training for context scaling. In *Thirty-seventh*  
812 *Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=s1FjXzJ0jy>.
- 814 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez,  
815 Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Infor-*  
816 *mation Processing Systems*, volume 30, 2017.
- 818 vLLM. Performance and tuning, 2024. URL <https://docs.vllm.ai/en/latest/models/performance.html#chunked-prefill>.
- 820 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi,  
821 Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language  
822 models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Ad-*  
823 *vances in Neural Information Processing Systems*, 2022. URL [https://openreview.net/forum?id=\\_VjQlMeSB\\_J](https://openreview.net/forum?id=_VjQlMeSB_J).
- 825 Yuhuai Wu, Albert Jiang, Jimmy Ba, and Roger Baker Grosse. {INT}: An inequality benchmark  
826 for evaluating generalization in theorem proving. In *International Conference on Learning Rep-*  
827 *resentations*, 2021. URL <https://openreview.net/forum?id=O6LPudownQm>.
- 829 An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li,  
830 Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang,  
831 Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jin-  
832 gren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin  
833 Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao,  
834 Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wen-  
835 bin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng  
836 Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu,  
837 Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL  
838 <https://arxiv.org/abs/2407.10671>.
- 839 Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov,  
840 and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question  
841 answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceed-*  
842 *ings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–  
843 2380, Brussels, Belgium, October–November 2018. Association for Computational Linguistics.  
844 doi: 10.18653/v1/D18-1259. URL <https://aclanthology.org/D18-1259>.
- 845 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R  
846 Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In  
847 *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=5Xc1ecx0lh>.
- 849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863

---

## A APPENDIX - PSEUDOCODE FOR CHUNKEDPREFILL AND WRITING IN THE MARGINS ALGORITHMS

---



---

### Algorithm 1: Inference with Chunked Prefill

---

```

864
865
866
867
868 Input : system_message (string)
869 context (string)
870 instruction (string)
871 llm (object)
872 Output: output (string)
873 1 context ← system_message + context;
874 2 segments ← split (context);
875 3 past_key_value ← [];
876 4 for segment ∈ segments do
877 | // add the segment to the KV cache
878 5 | prefill (llm, past_key_value, segment);
879 6 end
880 7 output ← generate (llm, past_key_value, instruction);
881 8 return output

```

---



---

### Algorithm 2: Writing in the Margins

---

```

883
884 Input : system_message (string)
885 context (string)
886 instruction (string)
887 extractive_summary_prompt (string)
888 classification_prompt (string)
889 llm (object)
890 Output: output (string)
891 1 context ← system_message + context;
892 2 segments ← split (context);
893 3 past_key_value ← [];
894 4 positive_margins ← [];
895 5 for segment ∈ segments do
896 | // add the segment to the KV cache
897 6 | prefill (llm, past_key_value, segment);
898 | // generate using the content of the KV cache and then
899 | // discard any
900 | // tokens added to the KV cache by the prompt and the
901 | // generated tokens
902 7 | margin ← generate (llm, past_key_value, extractive_summary_prompt);
903 8 | classification_input ← format (classification_prompt, margin, instruction);
904 9 end
905 | // do not use any past KV cache to classify
906 10 classification_result ← generate (llm, NULL, classification_input);
907 11 if classification_result = true then
908 | 12 | append (positive_margins, margin)
909 13 end
910 14 all_positive_margins ← concatenate (positive_margins);
911 15 prefill (llm, past_key_value, all_positive_margins);
912 16 output ← generate (llm, past_key_value, instruction);
913 17 return output

```

---

912  
913  
914  
915  
916  
917

	Hello	my	name	is	John	This	is	a	dog
Hello	1	0	0	0	0	0	0	0	0
my	1	1	0	0	0	0	0	0	0
name	1	1	1	0	0	0	0	0	0
is	1	1	1	1	0	0	0	0	0
John	1	1	1	1	1	0	0	0	0
This	0	0	0	0	0	1	0	0	0
is	0	0	0	0	0	1	1	0	0
a	0	0	0	0	0	1	1	1	0
dog	0	0	0	0	0	1	1	1	1

Figure 4: **Sequence packing.** Sequence packing allows to pack multiple unrelated documents in the same sequence. By adjusting the attention mask, we can avoid cross-contamination. This speeds up training time by reducing the number of padding tokens. A similar technique can also be used to inference from multiple prompts using the same sequence.

## B APPENDIX - DECOUPLING EXTRACTION AND CLASSIFICATION

Writing in the margins generates supplemental information by leveraging a partially prefilled KV cache. Each subsequent segment  $c$  in the KV cache can be used to generate an annotation, known as a “margin note”. To avoid providing the model with all the margins, we ask the model to generate the first token corresponding to the margin classes: relevant vs. irrelevant. In this section, we explore the possibility of decoupling the extraction and classification steps, which will allow for using separate prompting strategies. This separation might further boost the performance of the WiM pattern. We demonstrate that one can use the same instance of the model to perform both the computation of the margins and their classification.

In a naive implementation of such overlapped computation, the user may treat the classification request as an additional sequence and batch it with the prefilling request; this approach would require a very large number of padding tokens to align the two sequences. A more computationally efficient solution is to pack the classification request into the same sequence used to prefill the context and adjust the attention mask accordingly. An example of such a mask is provided in Figure 4. This technique is utilized during the pre-training of language models to reduce the number of padding tokens.

The first request to the language model would only contain the first segment  $c_1$  and the additional extractive instruction  $I_A$  (the “extractive summary prompt”). The attention mask at this point is provided in Figure 5 and Figure 6. This would generate the first margin  $M_0$ . After generating  $M_0$ , the instruction prompt  $I_A$  and all the subsequent tokens generated in  $M_0$  can be removed from the KV cache, leaving the KV cache only with  $c_1$ . In order to not grow or shrink a dynamically allocated KV cache, it is possible to use a static KV cache, as the number of total tokens in each segment,

972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

		$c_1$				$I_A$		PAD		
		K1	K2	K3	K4	K5	K6	P	P	...
$c_1$	Q1	1	0	0	0	0	0	0	0	0
	Q2	1	1	0	0	0	0	0	0	0
	Q3	1	1	1	0	0	0	0	0	0
	Q4	1	1	1	1	0	0	0	0	0
$I_A$	Q5	1	1	1	1	1	0	0	0	0
	Q6	1	1	1	1	1	1	0	0	0

Figure 5: **Prefilling of the first segment  $c_1$  along with the extractive instruction  $I_A$ .** Padding tokens are shown for clarity in case of a statically allocated KV cache, but they do not need to be attended to or used in the KV sequence when calculating the attention. The KV sequence should be a *slice* of the KV tensor that includes only non-padding tokens.

		$c_1$				$I_A$		$M_0$	PAD	
		K1	K2	K3	K4	K5	K6	K7	P	...
$M_0$	Q7	1	1	1	1	1	1	1	0	0

Figure 6: **Token generation using the prefilled KV cache.** Each generated token replaces a padding token in the KV cache.

extractive instruction and classification prompt is known in advance, so is the maximum number of tokens for each margin  $M_i$  and classification result  $\omega(I(M_i))$ .

Having generated the first margin  $M_0$ , it is possible to add the second segment  $c_2$  to generate the second margin  $M_1$  while at the same time classifying the previously generated margin  $M_0$ . To do so, the KV cache is prefilled with subsequent tokens  $c_2$ , the extractive instruction  $I_A$  and a number of padding tokens to accommodate the generated tokens of margin  $M_1$ . Moreover, the KV cache is also expanded by adding the classification instruction  $I(M_0)$  and a number of padding tokens to accommodate the generated tokens for the classification result  $\omega(I(M_0))$ . The attention mask at this point is provided in Figure 7.

Autoregressive token generation of the margin  $M_1$  and the classification result  $\omega(I(M_0))$  can be done in parallel by projecting the last token of each sub-sequence into logits. Each generated token can then be added in place of a padding token in each subsequence to generate successive tokens. Token generation at this stage is shown in Figure 8.

By using a statically allocated KV cache and by keeping track of how many tokens are used in it, it is possible to use a partial *view* (also known as “tensor slicing”) of the KV tensor without any computational overhead. It is also possible to use techniques like PagedAttention (Kwon et al., 2023) to allocate the KV cache block by block, in order to optimize the memory consumption while benefiting from a partial static allocation.

		$c_1$				$c_2$				$I_A$		PAD		$I(M_0)$			PAD		
		K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	P	...	K21	K22	K23	P	P	...
$c_2$	Q5	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Q6	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	Q7	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	Q8	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
$I_A$	Q9	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	Q10	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
$I(M_0)$	Q21	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	Q22	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
	Q23	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0

Figure 7: **Prefilling of the second segment  $c_1$  along with the extractive instruction  $I_A$ .** In this case the padding tokens between  $I_A$  and  $I(M_0)$  must be included in the KV sequence when calculating the attention to retain the memory continuity of the tensor, but the terminal padding tokens need not to. Each token in the second segment  $c_2$  needs to attend all tokens in the first segment  $c_1$ . The classification prompt  $I(M_0)$  be considered a completely separate document in the same sequence as prefiling.

		$c_1$				$c_2$				$I_A$		$M_1$	PAD	$I(M_0)$			$\omega(I(M_0))$	PAD	
		K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11	...	K21	K22	K23	K24	P	...
$M_1$	Q11	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
$\omega(I(M_0))$	Q24	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0

Figure 8: **Parallel token generation of the margin  $M_1$  and the classification result  $\omega(I(M_0))$ .** Each generated token replaces a padding token in its specific subsequence.

### C APPENDIX - PROMPTING

For all benchmarks, we respected their original formulation. In all cases, the prompt strategy for the Long Context LLM baseline could be expressed as:

```
{system_message}
```text
{context}
```
{instruction}
{query}
```

Where `system_message` and `instruction` were usually the task instructions split into two parts and appended before and after the main context respectively.

In the RAG approach, we used the original prompt but replaced `context` with all relevant segments concatenated by a newline sign.

In WiM inference, all constructed prompts shared the common prefix:

```
{system_message}
```text
{context}
```
```

This was necessary for the efficient reuse of the KV cache. To ensure that predictions were comparable, we manually identified a promising prompt for the margin generation and final prediction steps for all evaluated models.

#### C.1 MARGIN GENERATION

For each intermediate context  $\text{context}_i = \sum_1^i c_i$  and instruction  $I$ , we used the following extractive summary prompt  $I_A$  to generate a margin note  $M_i$ :

```

1080 I_A = ""
1081 {system_message}
1082 ```text
1083 {context_i}
1084 ```
1085 Copy over all context relevant to the query: {query}
1086 Provide the answer in the format: <YES/NO>#<Relevant context>.
1087 Here are rules:
1088 - If you don't know how to answer the query - start your answer with NO#
1089 - If the text is not related to the query - start your answer with NO#
1090 - If you can extract relevant information - start your answer with YES#
1091 - If the text does not mention the person by name - start your answer
1092 ↪ with NO#
1093 Example answers:
1094 - YES#Western philosophy originated in Ancient Greece in the 6th century
1095 ↪ BCE with the pre-Socratics.
1096 - NO#No relevant context.
1097 ""

```

1096 In our experiments, the margin generation step was combined with the classification step; the first  
 1097 token generated was a class label. We conditioned the generation of a margin based on the first token;  
 1098 i.e., we continued the generation only if the first token was YES. Additionally, the prompt included  
 1099 an explanation designed to enforce specific formatting and to prevent the model from inserting  
 1100 comments before delivering its judgment.

1101 In Appendix B, we explore the possibility of decoupling margin generation and classification  
 1102 prompts while using the same instance of the model.

## 1104 C.2 FINAL WIM PROMPT WITH ACCUMULATED MARGINS

1106 We used two variants of the prompt, depending on the number of retrieved margins.

### 1108 C.2.1 SINGLE MARGIN

```

1110 {system_message}
1111 ```text
1112 {context}
1113 ```
1114 I asked my assistant to read and analyse the above content page by page
1115 ↪ to help you complete this task. This is a margin note left on the
1116 ↪ last page:
1117 ```text
1118 QUERY: {query}
1119 ANSWER: {M_i}
1120 ```
1121 Read again the note(s) and the provided content, take a deep breath and
1122 ↪ answer the query.
1123 {instruction}
1124 {query}

```

### 1124 C.2.2 MULTIPLE MARGINS

```

1125 {system_message}
1126 ```text
1127 {context}
1128 ```
1129 I asked my assistant to read and analyse the above content page by page
1130 ↪ to help you complete this task. Those are margin notes left on each
1131 ↪ page:
1132 ```text
1133 Page 0:
1134 QUERY: {query}
1135 ANSWER: {M_i}

```

1134 Page 1:  
 1135 QUERY: {query}  
 1136 ANSWER: {M\_j}  
 1137 ...  
 1138 ...  
 1139 Read again the note(s) and the provided content, take a deep breath and  
 ↪ answer the query.  
 1140 {instruction}  
 1141 {query}

1142  
 1143 We replaced the term “segment” with “page” to more closely replicate the human practice of writing  
 1144 in the margins. In our experiments, there was no relationship between the order of the segments and  
 1145 the page numbers; this is left as an optional implementation detail.

1146  
 1147 **C.3 PROMPT USED WITH ACCURACY METRIC FOR SQUAD, HTOPOTQA AND**  
 1148 **MULTIHOP-RAG**

1149 Evaluate the following exam answer. I will provide you with the query,  
 1150 ↪ target answer(s) and the answer provided by the student.  
 1151 The student’s answer does not need to preserve the casing of the target  
 1152 ↪ answers, and slight variations in phrasing are acceptable, provided  
 1153 ↪ the meaning remains correct.  
 1154 Provide the answer in the format: <YES/NO>#<Explanation>.

1155 Here are the rules:  
 1156 - If the student’s answer is correct - start your answer with YES#  
 1157 - If the student’s answer is wrong or it is missing - start your answer  
 1158 ↪ with NO#

1159 Example answers:  
 1160

1161 QUERY: As of 2016, about what percentage of adults aged 18 years or older  
 1162 ↪ were overweight?  
 1163 TARGET: 40%, forty percent  
 1164 ANSWER: forty percent  
 1165 YES#The student’s answer is correct.

1166 QUERY: What is the value of p in  $24 = 2p$ ?  
 1167 TARGET: 12, 12.0  
 1168 ANSWER: five  
 1169 NO#The student’s answer is wrong.

1170 QUERY: What is the ‘Lotus principle’?  
 1171 TARGET: The so-called Lotus principle is that ‘restrictions upon the  
 1172 ↪ independence of States cannot therefore be presumed  
 1173 ANSWER: The Lotus principle is a horticultural technique developed in  
 1174 ↪ ancient Egypt for cross-pollinating lotus flowers with roses to  
 1175 ↪ create fragrant, floating gardens.  
 1176 NO#No, the student’s explanation is wrong.

1177 QUERY: {query}  
 1178 TARGET: {target}  
 1179 ANSWER: {answer}

1180  
 1181 **C.4 PROMPT USED WITH F1 METRIC FOR CWE**  
 1182

1183 Evaluate the following exam answer. The student’s task was to identify  
 1184 ↪ the most common words in text. I will provide you with the target  
 1185 ↪ answer(s) - 10 words - and the student’s response.  
 1186 The student’s answer does not need to preserve the casing of the target  
 1187 ↪ words.  
 1188 Provide your assessment in the format: <Positive>#<Negative>#<Explanation  
 1189 ↪ >.

1188  
1189 Here are the rules:  
1190 - <Positive> denotes the number of common words that the student has  
1191 ↪ correctly identified (i.e., those included in the target list).  
1192 - <Negative> represents the number of words the student has identified as  
1193 ↪ common but are incorrect (i.e., not included in the target list).  
1194 - If the student's answer is non-relevant, simply return: 0#10#The answer  
1195 ↪ is garbage.  
1196 - If the student's answer includes Python code, mention "python" in your  
1197 ↪ explanation.  
1198 - If the student provides the full answer twice, consider only the first  
1199 ↪ submission.

1199 Example answers:  
1200  
1201 TARGET: jealousy, access, chase, recess, head, animal, capon, hops,  
1202 ↪ unequaled, mozzarella  
1203 ANSWER: Based on the provided content and the margin notes, the 10 most  
1204 ↪ common words in the input are:\n\n1. chase (100 occurrences)\n2.  
1205 ↪ unequaled (99 occurrences)\n3. access (98 occurrences)\n4. hops (97  
1206 ↪ occurrences)\n5. animal (95 occurrences)\n6. jealousy (96  
1207 ↪ occurrences)\n7. mozzarella (94 occurrences)\n8. recess (93  
1208 ↪ occurrences)\n9. capon (91 occurrences)\n10. oxygen (91 occurrences  
1209 ↪ ),  
9#1#The student's answer contains 9 correct words out of 10 given.

1210 TARGET: seeder, workout, wafer, chronograph, liability, hovercraft,  
1211 ↪ nestling, alloy, netball, addicted  
1212 ANSWER: To find the 10 most common words in the given input, we need to  
1213 ↪ count the occurrences of each word and then sort them in descending  
1214 ↪ order. Here are the 10 most common words and their counts:\n\n1.  
1215 ↪ \"floodplain\" - 25 occurrences\n2. \"acetate\" - 24 occurrences\n3  
1216 ↪ . \"nondisclosure\" - 23 occurrences\n4. \"caterpillar\" - 22  
1217 ↪ occurrences\n5. \"moai\" - 21 occurrences\n6. \"video\" - 20  
1218 ↪ occurrences\n7. \"bacterium\" - 19 occurrences\n8. \"frog\" - 19  
1219 ↪ occurrences\n9. \"carnivore\" - 18 occurrences\n10. \"cattle\" - 18  
1220 ↪ occurrences\n\nNote: The input provided is a long list of words,  
1221 ↪ and the above list is based on the most frequent words found in the  
1222 ↪ list. The actual list may vary depending on the specific input.  
0#10#The student's answer provides only a generic answer and fails to  
1223 ↪ solve the problem.

1223 TARGET: jealousy, access, chase, recess, head, animal, capon, hops,  
1224 ↪ unequaled, mozzarella  
1225 ANSWER: To find the 10 most common words in the provided input, we can  
1226 ↪ use a Python script that reads the text, splits it into words,  
1227 ↪ counts the occurrences of each word, and then prints the top 10  
1228 ↪ most common words along with their counts. Here's a Python script  
1229 ↪ that accomplishes this:\n\npython\nfrom collections import Counter\  
1230 ↪ n\n# Read the text from the input\nwith open('input.txt', 'r') as  
1231 ↪ file:\n text = file.read()\n\n# Split the text into words\nwords =  
1232 ↪ text.split()\n\n# Count the occurrences of each word\nword\_counts =  
1233 ↪ Counter(words)\n\n# Get the 10 most common words\  
1234 ↪ nmost\_common\_words = word\_counts.most\_common(10)\n\n# Print the 10  
1235 ↪ most common words and their counts\nfor word, count in  
1236 ↪ most\_common\_words:\n print(word, count) The provided text\ntext =  
1237 ↪ 1.jealousy 2. gauge 3. work 4. townhouse 5. ubiquitous 6. regulator  
1238 ↪ 7. oxygen 8. verdict 9. war 10. verdict 11. rag 12. rag  
1239 1#11#The student's answer contains python code. One word is correct but  
1240 ↪ it contains also other 11 incorrect words.

1239 TARGET: {target}  
1240 ANSWER: {answer}  
1241