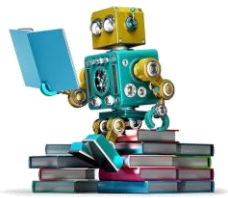
SDML
ML Paper Review

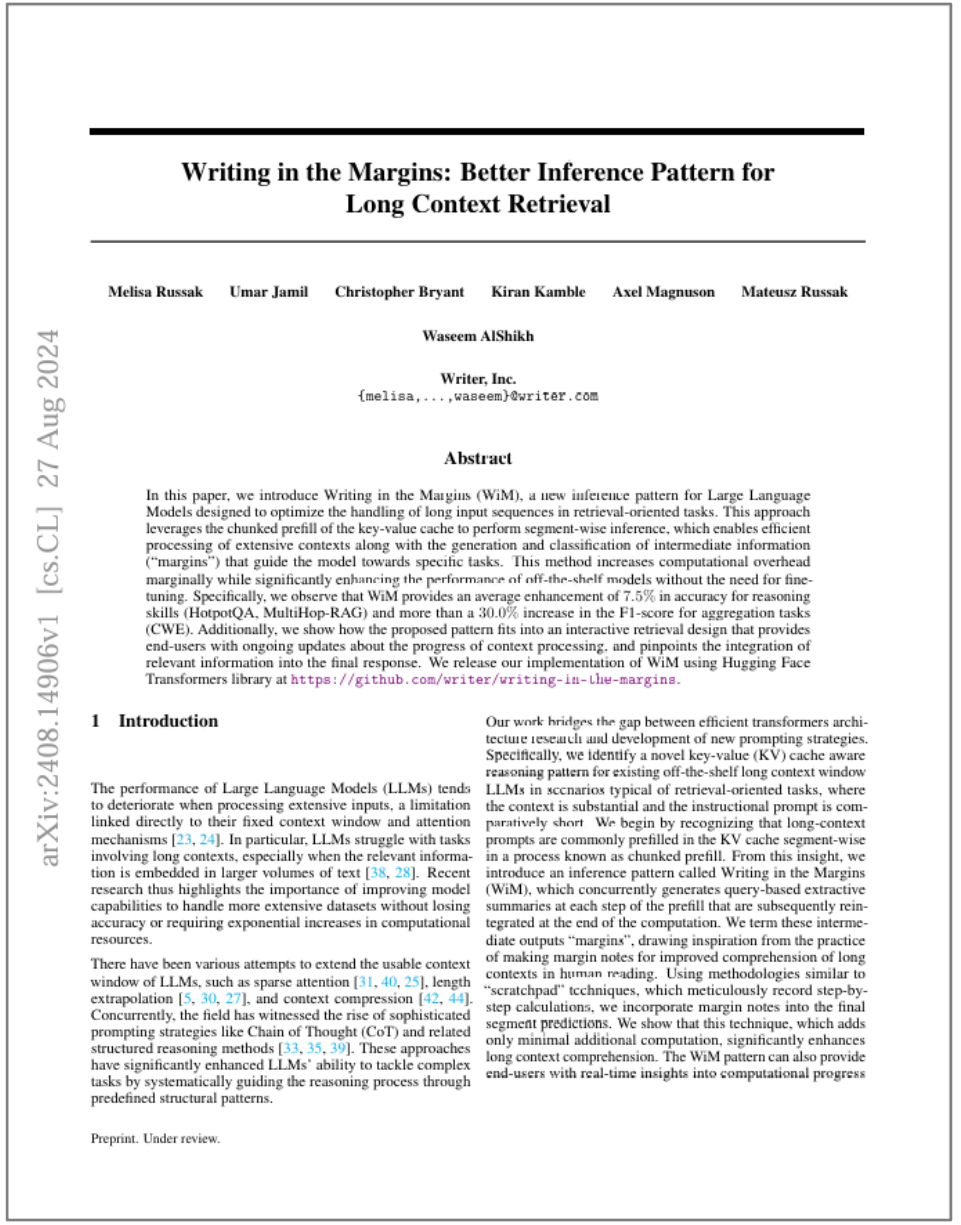
October 2024



Writing in the Margins: Better Inference Pattern for Long Context Retrieval

Melisa Russak et al., Writer, Inc.

https://www.arxiv.org/abs/2408.14906



Writing in the Margins: Better Inference Pattern for Long Context Retrieval

Melisa Russak Umar Jamil Christopher Bryant Kiran Kamble Axel Magnuson Mateusz Russak

Waseem AlShikh

Writer, Inc.
{melisa,...,waseem}@writer.com

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 more than 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 https://github.com/writer/writing-in-the-margins.

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 [23, 24]. In particular, LLMs struggle with tasks involving long contexts, especially when the relevant information is embedded in larger volumes of text [38, 28]. 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 [31, 40, 25], length extrapolation [5, 30, 27], and context compression [42, 44]. Concurrently, the field has witnessed the rise of sophisticated prompting strategies like Chain of Thought (CoT) and related structured reasoning methods [33, 35, 39]. 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 additional computation, significantly enhances long context comprehension. The WiM pattern can also provide end-users with real-time insights into computational progress

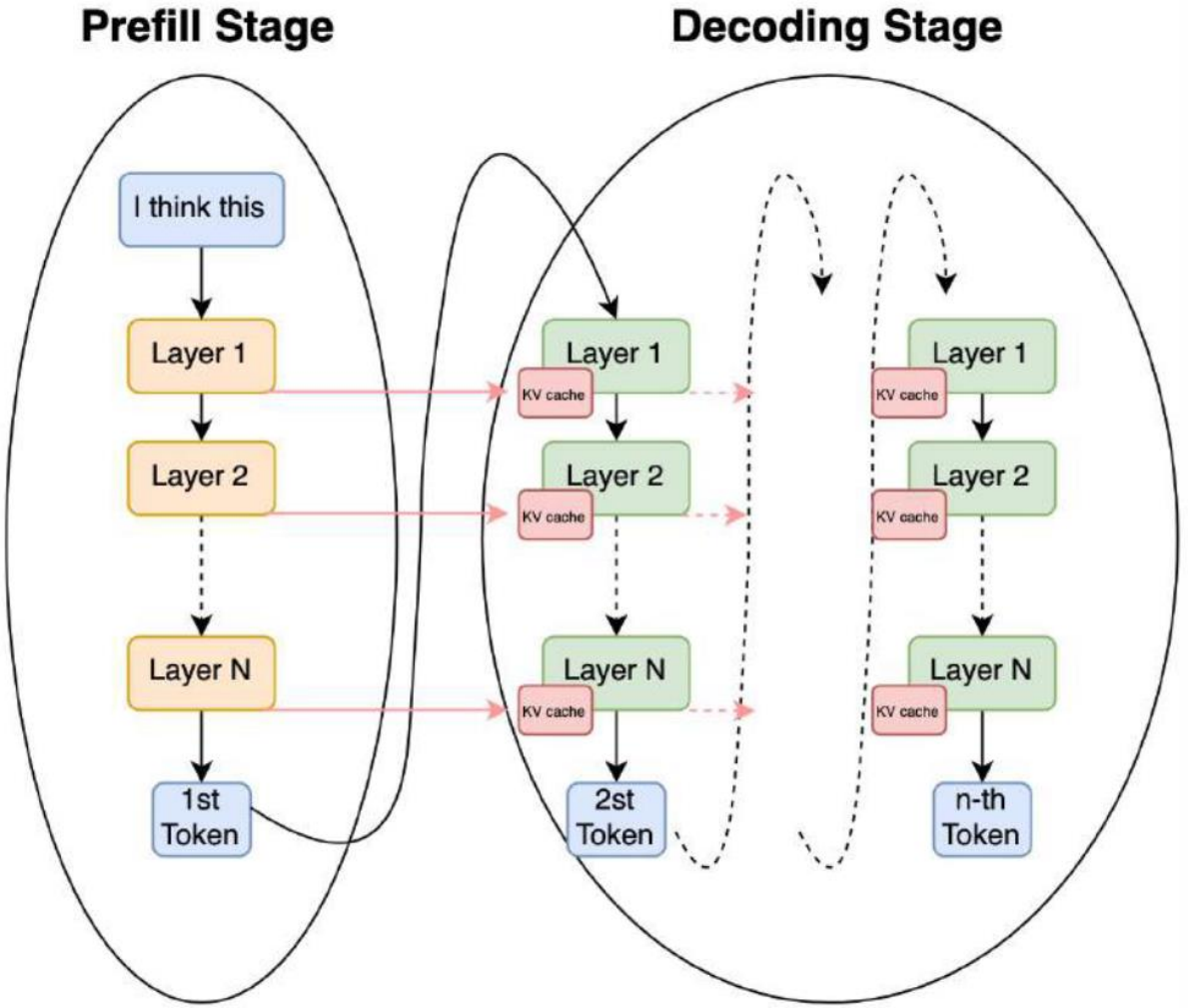
Preprint. Under review.

Writing in the margins overview

- Typical RAG workflow inserts selected snippets into the context for the LLM to attend to. For long contexts, performance can suffer.
- Here (WiM), they augment the text snippets with short summaries
  - They further optimize by only including summaries that are task relevant
- The clever engineering part is that they create these summaries in a way that adds little computational overhead
  - When processing the prompt and context in a decoder-only LLM, the keys and values for all these tokens need to be computed and saved in the KV cache
  - WiM appends a summarization prompt to each snippet and runs inference
  - They save the summary for later, but throw away the post-snippet KV data
  - Sneaking in summarization this way is much cheaper than a full inference pass

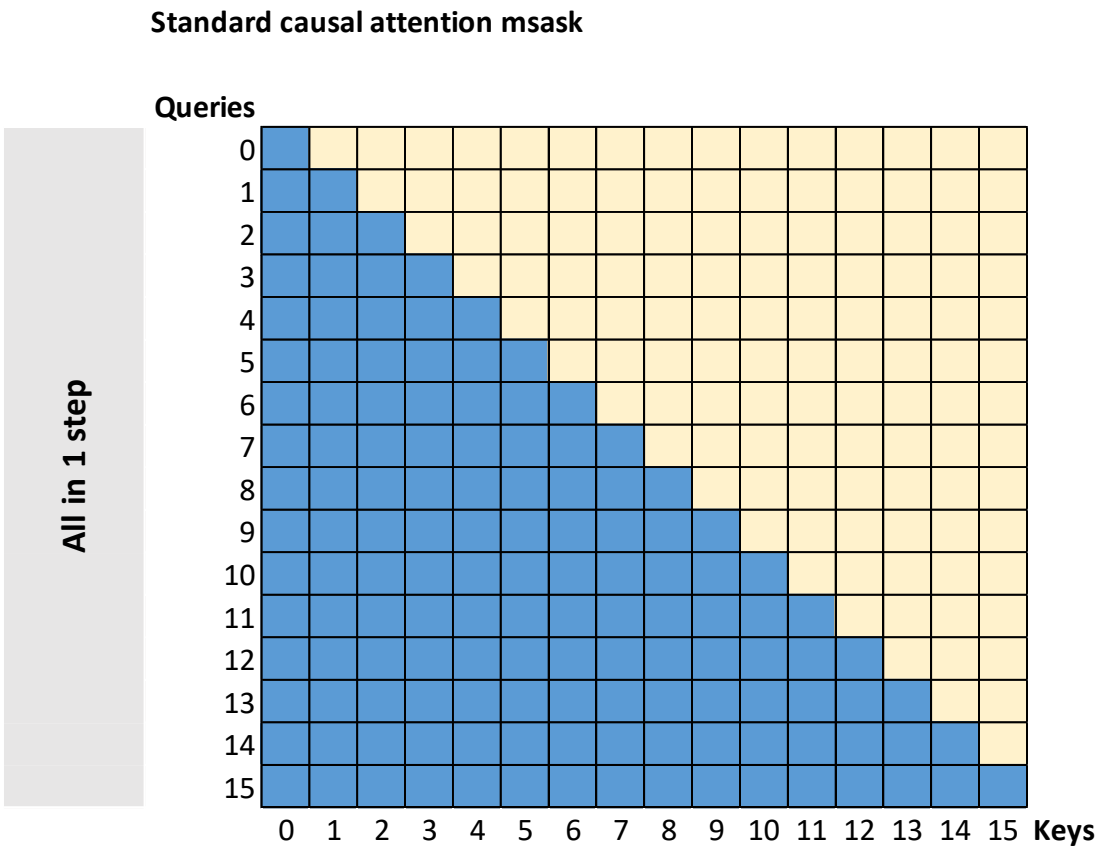
Prefill and decoding stages

- LLM inference happens in two very different stages
- During the prefill stage, we have many tokens from the prompt we can process together
- During decoding, we predict the next token, one token at a time



https://sites.google.com/view/kdd-2023-tutorial/home

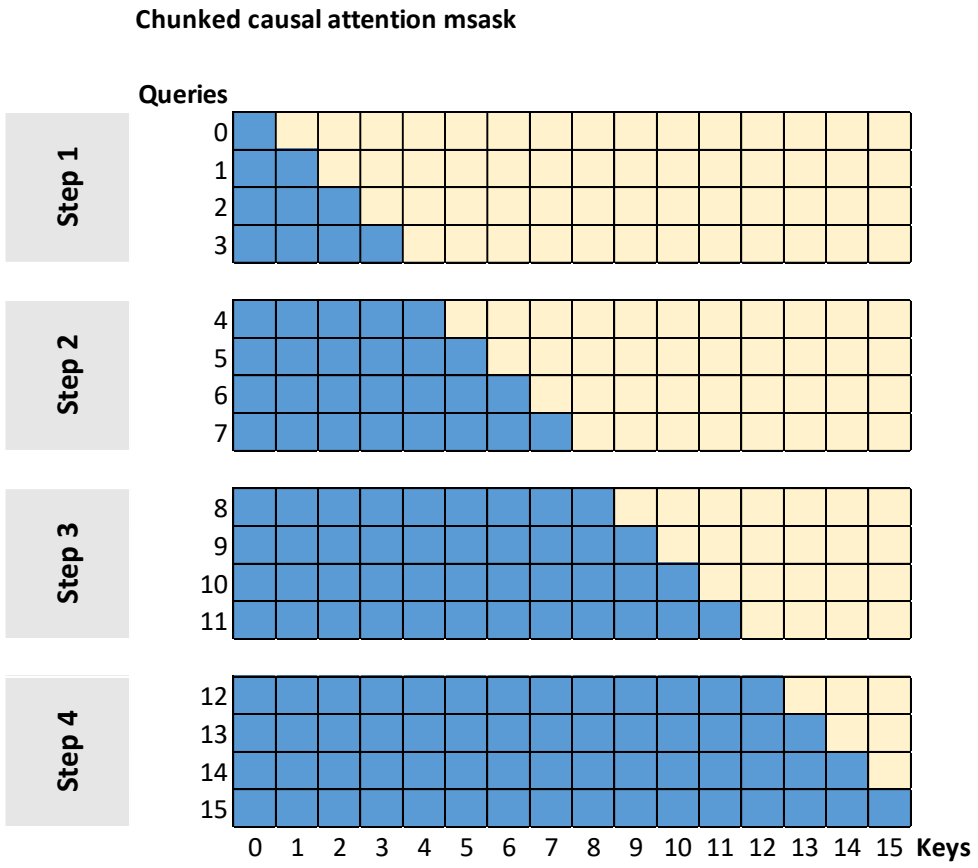
Standard attention



Blue squares are keys that each query is allowed to attend to
Yellow squares will have zero attention scores

- Every query, shown in rows, is multiplied with every key, shown in columns
- Keep only the blue squares
- In each position, the same query vector is multiplied with every one of the keys
- Likewise the same key vector values will be used with every one of the queries

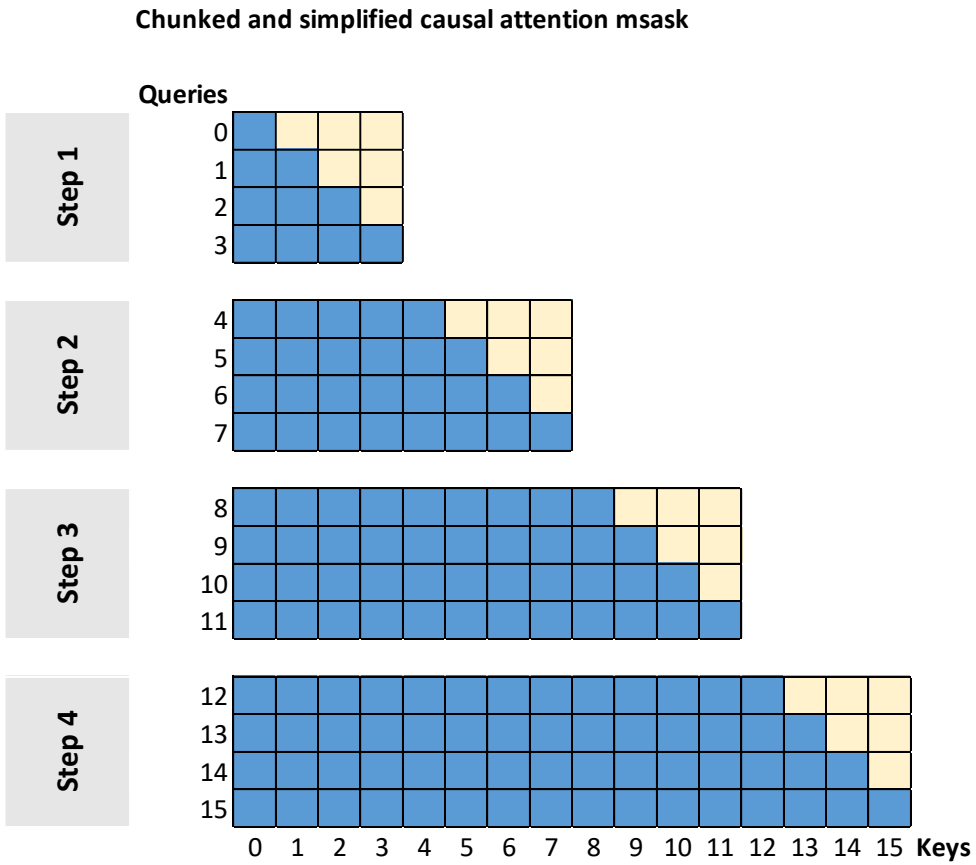
Chunked attention



Blue squares are keys that each query is allowed to attend to
Yellow squares will have zero attention scores

- Breaking the attention calc into chunks reduces size of each step
- It’s the same number of FLOPS
- But this reduces memory needed
- Concatenating the results of all steps results in the exact same answer

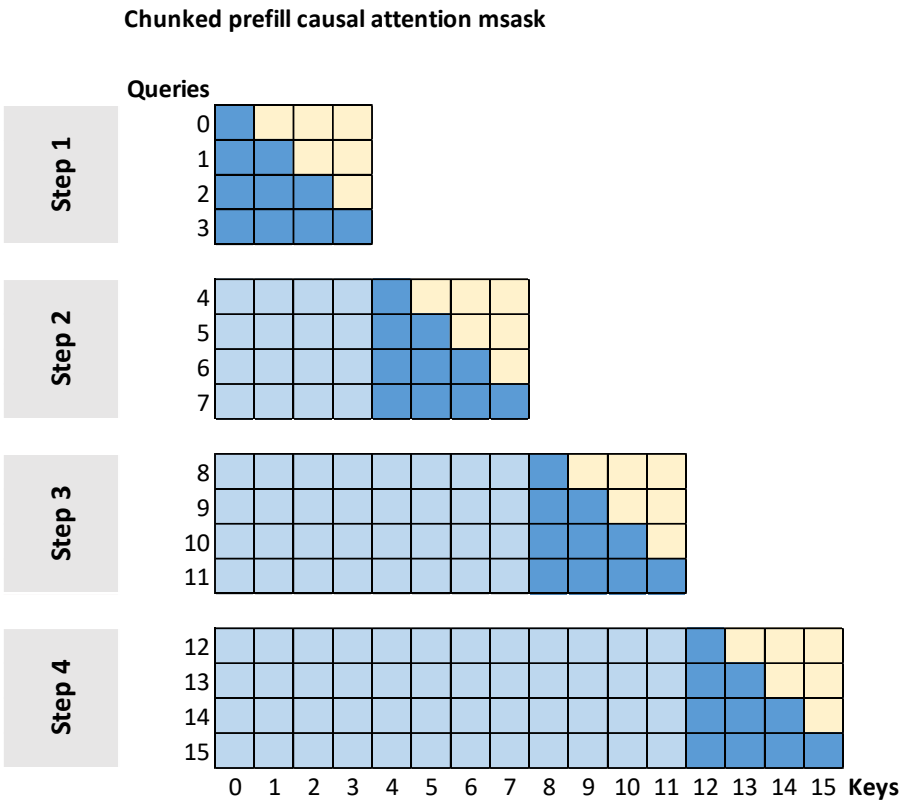
Simplified chunked attention



Blue squares are keys that each query is allowed to attend to
Yellow squares will have zero attention scores

- Since we are throwing away the results in yellow squares, we don't have to actually calculate them
- We can do these smaller matrix multiplies
- When concatenating the results, we need to pad zeros wherever we removed yellow squares

Chunked prefill attention



Blue squares are keys that each query is allowed to attend to

Dark blue squares calculate keys and values from scratch, and those are used in attention

Light blue squares reuse cached keys and values from previous steps, and those are used in attention

Yellow squares will have zero attention scores

- If we do the steps in order, from top to bottom, then the keys and values in light blue will have already been calculated by earlier steps
- We can cache and reuse them
- More importantly, we can continue to cache keys and values when decoding proceeds

Chunked Prefill

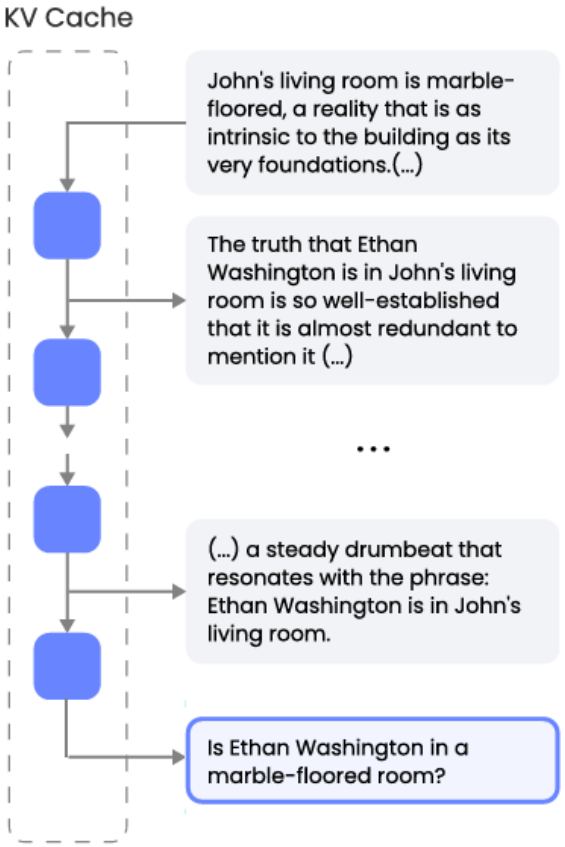
- Typical LLM inference consists of prefill phase then decoding phase
  - Prefill computes all keys and values for the prompt, storing in the KV cache
  - Decoding computes one next token at a time, using KV cache
- Naïve prefill of a long context of length L:
  - Performs attention on L queries and L keys, requiring $O(L^2)$ memory
  - It also can tie up the GPU for a long period of time, forcing other users to wait until the entire computation is finished
- Dividing up the prefill phase, called *chunked prefill* improves both:
  - Say you have N chunks of length K
  - Each chunk of K queries requires $O(KL)$ memory
  - After each chunk finishes, they system can decide if it wants to combine other users into the same batch as the next chunk, lowering response time

Writing in the margins method

- RAG is assumed to have long snippets of text retrieved, and a relatively short main prompt (such as question answering)
- During chunked prefill, when calculating the keys and values for a chunk, they don’t immediately move on to the next chunk
  - Instead, they append a small prompt asking the LLM to extract information from the text seen so far, with respect to the main prompt
  - They go farther, and ask the LLM to rate each summary for relevance to the main prompt
- After finishing normal prefill, they append the relevant summaries, then the original main prompt, and let decoding happen as usual

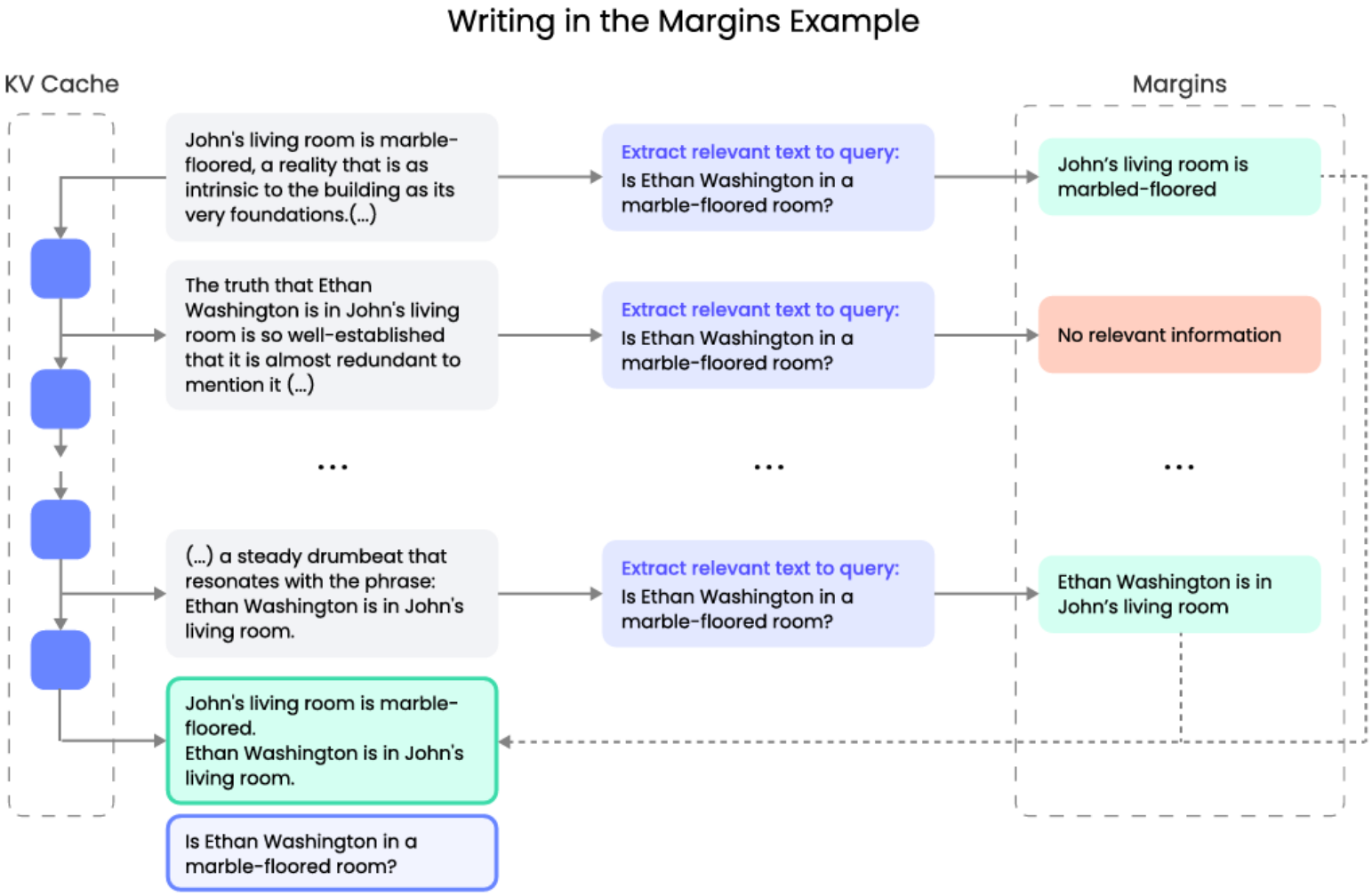
Standard RAG flow

- Main prompt is: “Is Ethan Washington in a marble-floored room?”
- RAG retrieves snippets of text
- This long context is broken into chunks (blue boxes) and the KV cache is populated
- The main prompt is appended
- Then, decoding runs



Example WiM flow

- Main prompt is: “Is Ethan Washington in a marble-floored room?”
- After chunked prefill for each blue box, a summary is extracted
- Each summary is also tested for relevance
- Good summaries are added right before the main prompt
- Then, decoding runs



WiM summary generation

- After each chunk, during chunked prefill, they append a small prompt asking the LLM to summarize the information so far, with respect to the main prompt
  - This summarization prompt is short, so inexpensive to prefill
- They run decode to get the short summary (the *margin*) from the LLM
  - This is fast, because most of the work needed is already in the KV cache
- The text of the summary is saved off for later use
- The keys and values from the summary prompt and decoding are thrown away
- From the perspective of the KV cache, it's as if the extra stuff never happened

WiM summary classification

- Each summary generated during chunked prefill is also classified as to whether or not the LLM found text relevant to the main prompt
- They could run a completely separate inference pass through the LLM, but the main results in the paper come from baking this into the summarization prompt
  - This entire process is fast because the summary and classification prompt are both very short

```
I_A = """
{system_message}
‘‘‘text
{context_i}
‘‘‘
Copy over all context relevant to the query: {query}
Provide the answer in the format: <YES/NO>#<Relevant
    ↪  context>.
Here are rules:
- If you don’t know how to answer the query - start
    ↪ your answer with NO#
- If the text is not related to the query - start
    ↪ your answer with NO#
- If you can extract relevant information - start
    ↪ your answer with YES#
- If the text does not mention the person by name -
    ↪ start your answer with NO#
Example answers:
- YES#Western philosophy originated in Ancient
    ↪ Greece in the 6th century BCE with the pre-
    ↪ Socratics.
- NO#No relevant context.
"""
```

WiM modified main prompt

- For the WiM technique, they added a little text to explain the *margins* that are being added to the context
- The prompt on the right shows how it explained margin notes from the user's assistant
- The instructions ask the LLM to read both the content and the *margin* notes, then respond to the main prompt

```
{system_message}
‘‘‘text
{context}
‘‘‘
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}
...
‘‘‘
Read again the note(s) and the provided content,
    ↪ take a deep breath and answer the query.
{instruction}
{query}
```

Evaluation

- Three tasks evaluated, with long context windows generated
  - Simulated irrelevant content filled up rest of long context
- Multi-hop question answering
  - Based on HotpotQA benchmark and the MultiHop-RAG benchmark
- Needle retrieval/Single-hop reasoning
  - Based on SQuAD benchmark
- Aggregation
  - Based on Common Words Extraction (CWE)

- Seven open LLMs w/128k context window support:
  - Phi-3-small-128k-instruct
  - Qwen2-7B-Instruct
  - Meta-Llama-3.1-8B-Instruct
  - Phi-3-medium-128k-Instruct
  - Palmyra-4-Chat-128K
  - Meta-Llama-3.1-70B-Instruct
  - Qwen2-72B-Instruct

Results

- SQuAD was too easy and RAG did best
- WiM consistently improved results for the HotpotQA, MultiHop RAG, and CWE experiments, especially as the context got larger

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

Ablations

- Results were worse without filtering for relevant *margin* content and if the *margins* replaced the snippets instead of being appended

model name	filtered (WiM)	all
Phi-3-small-128k-instruct	**0.58**	0.54
Qwen2-7B-Instruct	**0.65**	0.63
Meta-Llama-3.1-8B-Instruct	**0.70**	**0.70**
Phi-3-medium-128k-instruct	**0.65**	0.64
Palmyra-4-Chat-128K	**0.64**	0.55
Meta-Llama-3.1-70B-Instruct	0.72	**0.73**
Qwen2-72B-Instruct	**0.72**	0.71

Table 4: **Ablation: Filtering Margins** Classification and removal of irrelevant margins yields better results for majority of evaluated models. Results aggregated over HotpotQA, Multihop-RAG and SQuAD benchmarks.

model name	only margins	only context	both (WiM)
Phi-3-small-128k-instruct	**0.60**	0.55	0.58
Qwen2-7B-Instruct	0.62	0.56	**0.65**
Meta-Llama-3.1-8B-Instruct	0.68	0.68	**0.70**
Phi-3-medium-128k-instruct	0.57	0.58	**0.65**
Palmyra-4-Chat-128K	0.53	0.63	**0.64**
Meta-Llama-3.1-70B-Instruct	**0.72**	**0.72**	**0.72**
Qwen2-72B-Instruct	**0.72**	0.67	**0.72**

Table 5: **Ablation: Content Compression** Utilizing both margins and the entire context almost always maximizes the score despite the declining performance of the underlying model when faced with increasing input lengths. Results aggregated over HotpotQA, Multihop-RAG and SQuAD benchmarks.

Writing in the margins conclusion

- WiM is intended for the RAG problems with long retrieved text
- They designed a way to inexpensively extract small pieces of text relevant to the main prompt during the chunked prefill phase
  - This may be the beginning of other papers researching inexpensive prompts that can be answered through the use of the KV cache
- Only relevant text is used, and it appended to the retrieved text snippets, prior to the main prompt
- WiM showed good improvement on harder long context benchmarks like HotPotQA and MultiHop RAG
- They also mention that the *margin* summaries could be shown to the user as they are being created, providing the user more interactivity

References

- SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills
  Amey Agrawal et al. (2019)
  https://arxiv.org/abs/2308.16369
- Transformers Inference Optimization Toolset
  AstraBlog
  https://astralord.github.io/posts/transformer-inference-optimization-toolset/