Writing in Margins (WiM) with Reinforcement Learning to handle Long Context

Laukik Avhad lavhad@usc.edu
Shubham Gore spgore@usc.edu
Atharva Shembade shembade@usc.edu
Hruday Vuppala hvuppala@usc.edu
Priyanshi Doshi doship@usc.edu

Viterbi School of Engineering, University of Southern California, Los Angeles, California

Abstract

Large Language Models (LLMs) struggle with long input sequences, where relevant information is often buried in extensive context. The proposed approach extends the Writing in the Margins (WiM) method by integrating reinforcement learning (RL) to adaptively generate margin notes. This document outlines the background, a concise literature review, and a detailed work for developing an RL-enhanced WiM pipeline.

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing tasks, demonstrating remarkable capabilities in answering questions, summarization, and dialogue systems. However, one persistent challenge lies in their handling of long-context inputs, where relevant information may be scattered or buried within extensive textual passages. This limitation affects the model's ability to accurately comprehend and respond to user queries or prompts that rely on nuanced or distant contextual dependencies.

To address this issue, prior research has introduced the Writing in the Margins (WiM) [1] approach, a method that augments text with margins, context-sensitive annotations that help surface salient information. While promising, existing implementations of WiM rely on static heuristics or manually crafted strategies to determine how to generate margins and where to place them. These static methods may fail to adapt dynamically to different tasks or contexts, limiting their effectiveness. This project proposes an enhanced version of WiM by integrating Reinforcement Learning (RL) into the pipeline. The RL component is designed to learn optimal policies for generating margins based on feedback from downstream tasks, such as question answering or summarization. By treating margin generation as a sequential decision-making problem, the RL agent can adaptively determine what information to highlight, thereby improving the overall utility of long-context LLM processing.

The following sections explore the challenges LLMs face in handling long-context inputs, Related work, Methodology, experimental results, conclusion and future work.

2 Related Work

The challenge of effectively processing and utilizing information within long-context inputs is a significant hurdle for Large Language Models (LLMs). While LLMs have achieved remarkable performance across numerous NLP tasks, their ability to handle input sequences extending beyond a certain length is often limited by architectural constraints, primarily related to the quadratic complexity of the self-attention mechanism and memory limitations. This leads to "mid-sequence forgetting" .

To make information within long contexts more accessible to the model, the WiM injects context-sensitive “margins” into long inputs, acting as semantic breadcrumbs that steer the model toward the most relevant passages during inference. This strategy has boosted performance on tasks like question answering, but current systems depend on fixed, manually crafted annotation rules that don't easily adapt to new tasks or document formats.

Given that WiM shares similarities with RAG work on optimizing RAG pipelines using RL is highly pertinent. Researchers have applied RL to various stages:

- **Retrieval Decision**: Training agents to decide whether to trigger retrieval or rely solely on the LLM's parametric knowledge based on the query, potentially saving computational cost.[2] This relates to deciding if any margins are needed at all.
- **Query Formulation/Rewriting**: Using RL to train LLMs or smaller adapter models to

generate or rewrite user queries into more effective search queries that yield better results from the retriever.[3]

- **Preference Optimization**: Utilizing preference-based RL algorithms (DPO, KTO, IPO) to fine-tune components based on comparisons of outcomes derived from different retrieval strategies or retrieved sets.[4]
- **LLM-based Rewards**: Leveraging powerful LLMs (like GPT-4) as reference models to provide reward signals or preference labels for training RL agents, evaluating aspects like answer correctness or relevance.[2]

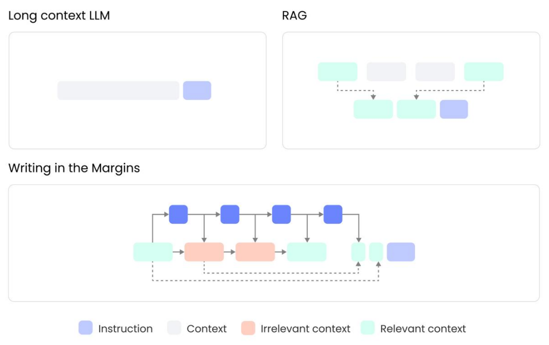
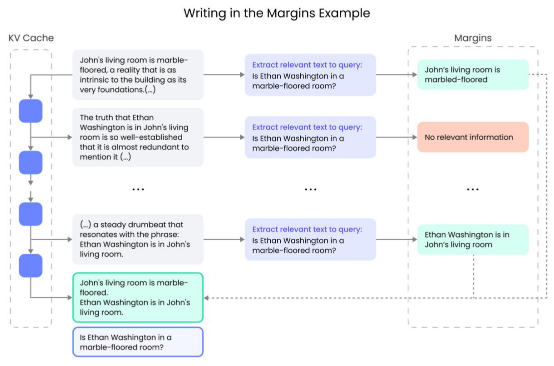


Figure 1: Figure 1: Prefilling KV cache using segments to generate margins


Writing in the Margins Example
KV Cache
Margins
John's living room is marble-floored, a reality that is as intrinsic to the building as its very foundations.(...)
Extract relevant text to query: Is Ethan Washington in a marble-floored room?
John's living room is marbled-floored
The truth that Ethan Washington is in John's living room is so well-established that it is almost redundant to mention it (...)
Extract relevant text to query: Is Ethan Washington in a marble-floored room?
No relevant information
...
...
...
(...) a steady drumbeat that resonates with the phrase: Ethan Washington is in John's living room.
Extract relevant text to query: Is Ethan Washington in a marble-floored room?
Ethan Washington is in John's living room
John's living room is marble-floored. Ethan Washington is in John's living room.
Is Ethan Washington in a marble-floored room?
Figure 2: Figure 2: Design Comparison LLM, RAG and WiM

3 Problem Description

Standard Transformer architectures, the foundation of most modern LLMs, face inherent limitations when dealing with long sequences. The primary bottleneck is the self-attention mechanism, which typically exhibits quadratic complexity in terms of computational cost and memory requirements with respect to the input sequence length. This makes naively scaling context windows computationally prohibitive. Beyond computational costs, LLMs often struggle to effectively utilize information spread across long contexts. Models perform significantly worse when relevant information is positioned in the central parts of the context, compared to the beginning or end.[5] Retrieval-Augmented Generation (RAG) offers a partial solution by retrieving relevant snippets from external knowledge sources instead of processing the entire source as context. However, RAG introduces its own set of challenges, including suboptimal retrieval quality (low precision or recall), difficulties in handling the retrieved noise, and integrating the retrieved information effectively with the generation process.

4 Methodology

4.1 Dataset

We use HotpotQA[9] dataset to train our model. It is a question-answering dataset that requires multi-hop reasoning across multiple documents. Each example includes a question,context, a supporting facts set, and a ground truth answer, encouraging models to synthesize information from different sources. It also includes supporting fact annotations, enabling explainable QA and intermediate supervision. We process this into query, context and supporting facts and answers as the ground truth for our model evaluation.

4.2 Data processing for WiM RL pipeline

4.2.1 Prompt decomposition

A standard prompt P consists of a context C followed by a final instruction I:

$$P = C + I$$

When $|C|$ exceeds the model's context window, naïve processing may truncate or forget early content.

4.2.2 Context chunking

We split the long context into N contiguous segments:

$$C = c_1 + c_2 + \ldots + c_N$$

where each c_i is a context of some fixed length.

4.2.3 Chunked prefill with PKV caching

For $i = 1 \dots N$, we feed chunk c_i (and any retained margin notes) into the transformer, appending its key/value (PKVs) to the cache. This simulates seeing the entire context without re-encoding from scratch.

4.2.4 Extractive margin generation

At each chunk boundary, an extractive instruction I_A is appended to guide the model to produce a short "margin" m_i:

$$[c_i + I_A] \to m_i$$

These intermediate summaries capture salient facts relevant to the final task.

4.3 Reinforcement Learning

In this project, Reinforcement Learning is used to enable adaptive and task-aware generation of margin notes within the WiM framework. Unlike static or heuristic-based annotation methods, our approach treats margin note generation as a sequential decision-making problem. At each step, the RL agent observes the textual context and decides whether, where, and what kind of margin note to insert.

We train this agent using Proximal Policy Optimization (PPO) [8], a policy-gradient algorithm that iteratively updates a stochastic policy to maximize expected return while enforcing a trust-region constraint via a clipped surrogate objective. Our reward function combines end-task performance with penalties for excessive or filtered-out notes, guiding the policy to produce only high-utility margins. PPO's stability and sample efficiency make it well suited for this setting, enabling the model to learn an effective margin-insertion strategy without destabilizing the base language model.

By integrating PPO-based RL into WiM, this project contributes a novel mechanism for dynamically optimizing margin annotations. We closely studied its principles and developed a customized version tailored to the specific requirements of our project. Our implementation draws inspiration from PPO's policy optimization strategies but adapts them to suit our data structure, reward signals, and computational constraints.

4.4 Model Training and Inference

We use a custom training and finetuned 500M parameter model on wikitext for our experiments.

4.4.1 Margin Generation

The `generate_rl_margin` method encapsulates how we invoke the base WiM pipeline to produce a candidate margin note for a given text segment:

- Prefills the KV cache with the segment text, recording how many tokens were consumed.
- Prefills again with the extractive summary prompt (e.g., "Summarize this for the query + segment"), capturing its output logits.
- Generates up to `max_new_tokens` of continuation via `generate_text_with_kv_cache`, using sampling (`top_p`, `temperature`) .
- Cleans the raw margin text (removing artifacts or trimming).
- Shrinks the KV cache back to its original size (or to right before the prompt), so that subsequent calls start from a known state.

This routine is called for every chunk that the RL policy decides to summarize.

4.4.2 Model Training

In our `train` method, we repeat the following over episodes:

Chunking Convert the full document into fixed-size segments, a hyperparameter which can be tuned.

Margin Sampling

- For each segment, clear the KV cache.
- Call generate_rl_margin.
- Compute reward for each margin generated
- Collect the sampled margins into a list.

PPO Update

- Build training examples by concatenating each segment + prompt + generated margin.
- Tokenize and mask out the context portion so only margin tokens contribute to the language modeling loss.
- For a fixed number of PPO epochs:
  - Compute model loss (cross-entropy on the margin tokens).

- Compute KL divergence between the policy and a frozen reference model.
- Add reward term (negative reward, since optimizers minimize).
- Sum these into a single loss, backpropagate (with gradient clipping), and step the optimizer.

4.4.3 Reinforcement Learning Formulation

- **State:** Implicit, encoded by how the policy model's input is constructed (segment embedding + prompt).

- **Action:** Generating a margin for every chunked segment.

- **Reward:** Rewarding the model for higher margin with higher score and penalizing it for lesser scores. Computed using supporting facts and query terms.

- **Margin Penalty Signal:** We leverage the supporting facts provided in the HotpotQA dataset alongside the key terms extracted from the query to assess the relevance of each generated margin. Margins that fail to align with these references incur a penalty, discouraging the inclusion of irrelevant or redundant summaries.

KL Divergence and Reference Model To prevent policy drift, we include a KL regularization term:

$$L_{KL} = \beta \cdot \mathrm{KL}[\pi_{policy}(\cdot \mid s) \parallel \pi_{ref}(\cdot \mid s)]$$

- π_{ref}: A frozen copy of the fine-tuned LLM (reference model).

- π_{policy}: The same architecture, updated during PPO.

- β: A hyperparameter from that controls the trade-off between exploration and staying close to the original LM outputs.

By jointly minimizing the language modeling loss, the KL term, and the negative reward, the policy learns to propose margins that are both high-quality and effective.

4.4.4 Inference

After training, our final system (RL-WiM) works as follows for a given test document and question: The document is streamed through the model chunk by chunk. At each potential break point, the RL policy network (now fixed) observes the current state and decides whether to create a margin. If reward margin generated is above a certain threshold, it appends the margin to the final prompt. If not, the model continues to the next chunk seamlessly. This continues until the document is exhausted. Finally, the model generates an answer to the question, with all the collected margins in its context.

5 Experimental Results

Experimental setup, including datasets, baseline methods, and evaluation protocols. We use a 3 prompt strategy using context, query, answer, generated-answer to evaluate our baseline model on the multiHop HopPotQA dataset. WimRL similar to WiM improves multihop reasoning as relevant margins are provided. The below results are captured over 500 datapoints in the HopPotQA dataset.

BERT SCORE	WiM + RL	Baseline Model
BERT-F1(Avg)	0.5	0.47
BERT-Precision(Avg)	0.6	0.4
BERT-Recall(Avg)	0.3	0.3

Table 1: Results on different models

6 Conclusions and Future Work

The "Writing in the Margins" (WiM) methodology presents an innovative inference-time strategy for improving the ability of Large Language Models to handle long context retrieval tasks, demonstrating notable performance gains without requiring model fine-tuning. Reinforcement Learning formulates margin generation and/or selection as sequential decision-making problems. WiM RL significantly improves long context handling compared to the baseline model.

For future work, optmization can be done by focussing on segment size for specifc models. WimRL method can be tried on bigger/larger models with larger context windows to achieve greater accuracy. A sliding window style processing of segment can be tested specifically for training on segments for margin generation without losing information.

7 Individual Contributions

Shubham Gore

- Designed state/action/reward structure and conducted initial reward function tuning.
- Implemented Margin reward model, KL divergence for PPO, chunked prefill and margin classification.

Atharva Shembade

- Fine-tuned the TinyLLama, LLaMA 3.2B model using LoRA, applied 4-bit quantization using BitsAndBytes, and optimized memory settings to run experiments on limited hardware.
- Worked on evaluation of base model and WimRL model using BERT scores (F1, precision, recall).

Laukik Avhad

- Implemented WiM baseline, including chunked prefill, margin generation logic, and integration of key-value cache.
- Tuned PPO hyperparameters (clip range, learning rate) and analyzed training stability.

Hruday Vuppala

- Dataset preparation, including preprocessing of HotpotQA and MultiHop-RAG, chunk segmentation, and aligning ground-truth answers.
- Assisted in designing margin relevance filtering and prompt formatting for training.

Priyanshi Doshi

- Implemented margin generation using PPO and fine-tuned parameters for WiM inference model.
- Formatted the training examples in the format context, query and answer for faster training.

References

[1] Russak, Melisa, Umar Jamil, Christopher Bryant, Kiran Kamble, Axel Magnuson, Mateusz Russak, and Waseem AlShikh. "Writing in the Margins: Better Inference Pattern for Long Context Retrieval." arXiv preprint arXiv:2408.14906 (2024).

[2] Kulkarni, Mandar, Praveen Tangarajan, Kyung Kim, and Anusua Trivedi. "Reinforcement learning for optimizing rag for domain chatbots." arXiv preprint arXiv:2401.06800 (2024).

[3] Chen, Guoxin, Minpeng Liao, Peiying Yu, Dingmin Wang, Zile Qiao, Chao Yang, Xin Zhao, and Kai Fan. "C-3PO: Compact Plug-and-Play Proxy Optimization to Achieve Human-like Retrieval-Augmented Generation." arXiv preprint arXiv:2502.06205 (2025).

[4] Xu, Ran, Wenqi Shi, Yuchen Zhuang, Yue Yu, Joyce C. Ho, Haoyu Wang, and Carl Yang. "Collab-RAG: Boosting Retrieval-Augmented Generation for Complex Question Answering via White-Box and Black-Box LLM Collaboration." arXiv preprint arXiv:2504.04915 (2025).

[5] Chaudhury, Subhajit, Payel Das, Sarathkrishna Swaminathan, Georgios Kollias, Elliot Nelson, Khushbu Pahwa, Tejaswini Pedapati, Igor Melnyk, and Matthew Riemer. "EpMAN: Episodic Memory AttentioN for Generalizing to Longer Contexts." arXiv preprint arXiv:2502.14280 (2025).

[6] Hosseini, Peyman, Ignacio Castro, Iacopo Ghinassi, and Matthew Purver. "Efficient solutions for an intriguing failure of llms: Long context window does not mean llms can analyze long sequences flawlessly." arXiv preprint arXiv:2408.01866 (2024).

[7] https://github.com/writer/writing-in-the-margins

[8] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms." arXiv preprint arXiv:1707.06347 (2017).

[9] Yang, Zhilin, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. "HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering." Conference on Empirical Methods in Natural Language Processing (EMNLP) (2018).