

# LeanContext: Cost-Efficient Domain-Specific Question Answering Using LLMs

Md Adnan Arefeen<sup>1,2 \*</sup>, Biplob Debnath<sup>1</sup>, and Srimat Chakradhar<sup>1</sup>

<sup>1</sup>NEC Laboratories America, <sup>2</sup> University of Missouri-Kansas City  
aa4cy@umsystem.edu, {biplob, chak}@nec-labs.com

## Abstract

Question-answering (QA) is a significant application of Large Language Models (LLMs), shaping chatbot capabilities across healthcare, education, and customer service. However, widespread LLM integration presents a **challenge for small businesses due to the high expenses of LLM API usage**. Costs rise rapidly when domain-specific data (context) is used alongside queries for accurate domain-specific LLM responses. One option is to summarize the context by using LLMs and reduce the context. However, this can also filter out useful information that is necessary to answer some domain-specific queries. In this paper, we shift from human-oriented summarizers to AI model-friendly summaries. Our approach, **LeanContext**, efficiently extracts  $k$  key sentences from the context that are closely aligned with the query. The choice of  $k$  is neither static nor random; we introduce a reinforcement learning technique that **dynamically determines  $k$  based on the query and context**. The rest of the less important sentences are reduced using a free open source text reduction method. We evaluate LeanContext against several recent **query-aware and query-unaware** context reduction approaches on prominent datasets (arxiv papers and BBC news articles). Despite cost reductions of 37.29% to 67.81%, LeanContext's ROUGE-1 score decreases only by 1.41% to **2.65% compared to a baseline that retains the entire context** (no summarization). Additionally, if free pretrained LLM-based summarizers are used to reduce context (into human consumable summaries), LeanContext can further modify the reduced context to enhance the accuracy (ROUGE-1 score) by 13.22% to 24.61%.

## Introduction

In recent times, large language models (LLMs) have seen extensive utilization, especially since the introduction of LLM APIs for customer-oriented applications on a large scale (Liu et al. 2023). These applications include chatbots (like GPT-4), language translation (Jiao et al. 2023), text summarization (Luo, Xie, and Ananiadou 2023; Yang et al. 2023; Zhang, Liu, and Zhang 2023a), and question-answering (QA) tasks (Tan et al. 2023), personalized robot assistance (Wu et al. 2023). While the zero-shot performance of the LLM model is nearly on par with fine-tuned models for specific tasks, it has limitations. One significant limitation is its **inability to answer queries about re-**

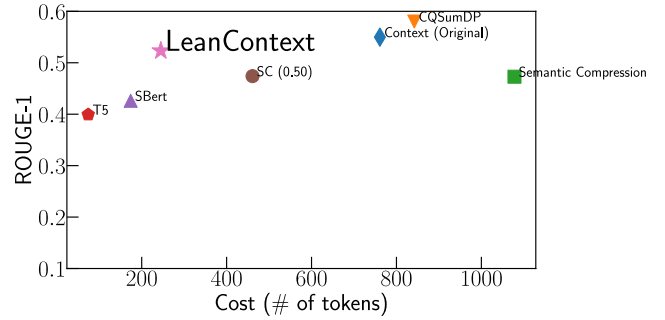


Figure 1: Compared to the original context LeanContext only drops in ~2% ROUGE-1 score with ~ 68% savings on BBCNews dataset (Li 2023).

**cent events on which it has not been trained**. This lack of exposure to up-to-date information can lead to inaccurate responses, particularly for domain-specific information processing, where the LLMs may not grasp new terminology or jargon. To build an effective domain-specific question-answering system, it becomes essential to educate the LLMs about the specific domains, enabling them to **adapt and understand new information accurately**.

LLMs can learn domain-specific information in two ways, (a) via *fine-tuning* the model weights for the specific domain, (b) via *prompting* means users' can share the contents with the LLMs as input context. Fine-tuning these large models containing billions of parameters is expensive and considered impractical if there is a rapid change of context over time (Schlag et al. 2023) e.g. a domain-specific QA system where the documents shared by users are very recent and from different domains. A more practical way is to select the latter approach i.e. the prompt-based solution, where relevant contents from user documents are added to the query to answer based on the context. Motivated by this, **we focus on prompt-based solutions for document-based QA systems**.

One of the challenges for long document processing using a prompt-based solution is the input prompt length being limited to a maximum length defined by the LLM API. The token limit of GPT-3.5 and GPT-4 vary from 4,096 to 32,768 max tokens limit proportional to the usage cost. Therefore, LLMs will fail to answer the query if the prompt length is

\*Work done during internship at NEC Laboratories America

larger than the **max token limit** due to the larger context length in the prompt. One suitable way to get rid of this problem is via **document chunking** (Harrison Chase 2022). In this case, initially, the user documents are segmented into chunks. Only the relevant chunks of fixed size are retrieved as context based on the query.

The cost for context-based querying using LLM APIs via prompting is associated with a cost that is proportional to the number of input tokens (contributing prompt cost), and the number of output tokens (contributing generation cost). According to a recent study (GPT-3 Cost), with 15,000 visitors having 24 requests per month, the cost of using GPT-3 (Davinci model) is \$14,400 per month (assuming prompt tokens = 1800, output tokens = 80) which is challenging for a small business to operate. For GPT-4, the cost is even higher than this amount. In this paper, **our focus is to reduce this cost.**

To mitigate the cost of using LLM API, the number of tokens in the context should be reduced as the cost is proportional to the length of the context. **A low-cost option to reduce the context is to summarize the context using free open-source summarizer models.** However, for domain-specific QA applications, the **pre-trained open-source summarizers do not contribute to good accuracy.** On the contrary, using a pay-per-use model like ChatGPT further increases the query processing cost instead of reducing it as the additional cost is added at the time of text reduction.

To this end, **we propose a domain-specific query-answering system LeanContext**, where users ask queries based on a document. To answer a query, **LeanContext first forms a context from the document based on the query by retrieving relevant chunks.** Next, it identifies the top- $k$  sentences related to the query from the context. LeanContext introduces a reinforcement learning technique that **dynamically determines  $k$  based on the query and context.** Then, LeanContext reduces the rest of the sentences in fragments by an open source text reduction method. Next, it **forms a new context by stitching top- $k$  sentences and reduced fragments** in the order of their appearance order in the original context. Finally, it invokes an LLM (like ChatGPT) to answer the query using that new context. It is to be noted that the goal of LeanContext is contrary to the summarization task that generates a meaningful summary for human users. Rather, in LeanContext, the reduced context will be consumed by a question-answering model like ChatGPT. Figure 1 shows the scenario of LeanContext, reducing the context size with accuracy close to the original context and outperforming other open-source models.

In summary, LeanContext makes the **following contributions:**

- It presents a low-cost domain-specific QA system, which reduces the LLM API usage cost by reducing the domain context through the selection of important sentences related to the query and keeping them intact, while reducing rest of the sentences in between important sentences through a free open-source summarizer. It proposes a reinforcement learning technique to select the percentage of the important sentences.

- It reduces the LLM API usage cost by 37.29% ~ 67.81% of a domain-specific QA system with little drop in performance by only 1.41% ~ 2.65%. (Table 1, Table 2).
- It boosts the QA performance by 13.22% ~ 24.59% (Table 3) by combining query-aware top- $k$  sentences with the reduced context generated through free open-source text summarizers.

## Related Work

For domain-specific tasks, LLMs can be utilized to adapt the domains **without modifying their inner parameters via discrete prompting** where distinct instructions with contexts can be delivered as an input to generate responses for downstream tasks (Ling et al. 2023; Brown et al. 2020). For domain-specific QA tasks, the domain can be reduced by context summarization to reduce the LLM cost. A lot of research has been conducted for summarizing text (Miller 2019; Yang et al. 2023). Existing research works can be categorized into two main parts: **(a) extractive and (b) abstractive.** The **extractive summarizers** (Miller 2019) first identify important sentences from the text, and next summarizes them. While **abstractive summarizers** (Laskar et al. 2023) reduce the context by generating new sentences. **The main goal of both approaches is to generate a meaningful summary for human users.** In contrast, the goal of LeanContext is to reduce context which will be **consumed by a question-answering model like ChatGPT.** For the prompt-based summarization task, recently, iterative text summarization (Zhang, Liu, and Zhang 2023b) has been proposed to refine the summary task in a feedback-based iterative manner. In aspect of query-based summarization (Yang et al. 2023) summaries are generated based on a domain set of specific queries.

Query-unaware text compression via prompting is also observed in recent literature. Semantic compression (Gilbert et al. 2023) involves generating systematic prompts to reduce context using ChatGPT model (GPT-3.5-turbo, GPT-4) and acquire reasonable compression compared to the zlib compression method. Due to limited context window size, recent literature focus on prompt context filtering. In selective context (Li 2023), token, phrase, or sentence-level query-unaware content filtering is proposed using the entropy of GPT-2 model logits for each entity. Usage of ChatGPT model in the medical domain especially in radiology is explored via prompt-engineering (Ma et al. 2023) to summarize difficult radiology reports. Extract-then-generate pipeline-based summarization improves abstractive summary faithfulness (Zhang, Liu, and Zhang 2023a) through the chain of thought (CoT) (Wei et al. 2022) reasoning. To reduce the cost of the use of LLM, FrugalGPT (Chen, Zaharia, and Zou 2023) proposed several ideas regarding prompt adaptation by query concatenation, LLM approximation by fine-tuning or caching, and LLM cascade by the selective selection of LLMs from lower to higher cost based on a query. Still, it lacks context compression ideas to reduce the prompt tokens.

It is to be noted that recent studies either focus on summarization as a downstream task or utilize the summary of

the context for the question-answering task. For most of the existing content filtering approaches, the main focus is to query-agnostic filter content with the deletion of less informative content or for solely summarization tasks. Using LLM for query-aware context reduction adds extra overhead for using pay-per-use LLM to answer correctly. In addition, in recent articles, the chunk-based preprocessing of the article is ignored by assuming each content in the dataset as a chunk. In LeanContext, the main focus is to reduce the LLM cost by considering query-aware context reduction. Due to the possibility of rapid change of domain-specific user data, fine-tuning LLM or parameter-efficient LLM is not a feasible solution. Utilizing open-source LLM (Touvron et al. 2023; Chung et al. 2022) model either does not perform well on domain data or adds additional deployment cost. Consider a small business to run, we consider using pay-per-use LLMs such as OpenAI LLMs to make the system running at a reasonable cost by reducing the context.

## Domain-specific QA System

In a domain-specific QA system, a context with a query is given to an LLM to get the answer. If the context size exceeds the max-token limits of the LLM API, the LLM will fail to answer the query. As a result, for long document processing, a vector database (ChromaDB; Pinecone) is used to store domain-specific documents into a number of small chunks so that a subset of relevant domain context can be retrieved from the long document rather than the whole document as context. A domain-specific QA system is shown in Figure 2. The QA system can be divided into two steps.

**(a) Domain data ingestion:** In this step, the documents,  $\mathcal{D}$  will be split into a number of fixed chunks ( $c$ ) by a text splitter. An embedding function computes the embeddings of each chunk using an embedding generator. The chunks along with the embedding vector ( $\mathbf{v}_c$ ) of each chunk are stored in a vector database.

**(b) QA:** At the question-answering (QA) step, given a user query  $q_i$ , similar  $N$  chunks are retrieved by their embeddings similar to query embedding using *semantic search*. These chunks form the context ( $\mathcal{C}$ ). Finally, the context which is a subset of domain data is fed into LLM to get the answer.

As domain-specific data and user queries are dynamic in nature, retrieving minimal context based on a query is challenging. One possible way is to make the chunk size and number of chunks dynamic so that the context contains minimal sentences to answer the query. But this solution is infeasible as the vector database needs to be reconfigured again per query with the change of domain. Instead, it will be more practical if **after getting the possible chunks as context, the context is further reduced based on a query to get the near-optimal cost of LLM**. Following this notion, we propose LeanContext, an adaptive context reduction system to reduce the prompt cost of ChatGPT like LLMs.

## LeanContext

The objective of LeanContext is to further reduce the context  $\mathcal{C}$  to  $\mathcal{C}'$  where the token count of  $\mathcal{C}'$  is smaller than the

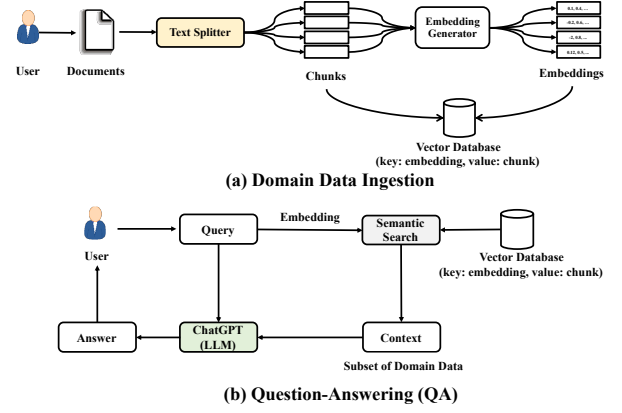


Figure 2: Workflow of a domain-specific QA system.

token count of  $\mathcal{C}$ . As LLM prompt cost is proportional to the token count of context, LeanContext helps to reduce the prompt cost of LLMs. In other words, if the total number of tokens in  $\mathcal{C}$  is  $T$  and the number of tokens in  $\mathcal{C}'$  is  $t$  then LeanContext reduces the token ratio  $\tau$  is defined as,  $\tau = \frac{t}{T}$  without compromising the accuracy ( $acc$ ) of the QA system. So, if the optimal accuracy of the system is  $acc^*$ , we formulate the **optimization problem of LeanContext** as follows.

$$\min. (1 - \alpha) \times \tau + \alpha \times |acc - acc^*| \quad (1)$$

Finally, the reduced context  $\mathcal{C}'$  and the query ( $q_i$ ) are given to the pay-per-use LLM API to answer the query. Then, the answer is shown to the respective user via an interactive interface.

For context-based QA, generally, the answers reside within a couple of sentences. If the smallest amount of context for a certain question can be identified, the same response can be provided by LLMs at a lower cost i.e. less prompt tokens. So, identifying the top- $k$  sentences can reduce the context without compromising accuracy. Motivated by this simple idea, we propose LeanContext which is shown in Figure 3.

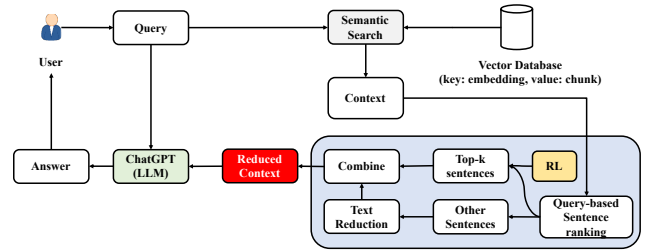


Figure 3: LeanContext System

After forming the context with semantic search, LeanContext first *rank*s the sentences of the context based on a query. Assuming the context consists of a sequence of sentences  $(s_1, s_2, s_3, \dots, s_n)$ , it extracts top- $k$  sentences similar to the query from context using the **cosine-similarity** function. To accomplish this, it computes the embedding of the

query ( $\mathbf{v}_q$ ), and the embedding is compared with each of the sentence embedding ( $\mathbf{v}_{s_i}$ ) and *top-k sentences* are identified.

$$\text{Top-}k \text{ sentences} = \text{sort}(\mathbf{V}, \text{similarity\_score}(\mathbf{v}_q, \mathbf{v}_{s_i}))$$

Here,  $1 \leq i \leq n$ . Using only top- $k$  sentences as a reduced context is a lightweight approach to reduce the cost LLM API usages. However, LeanContext boosts accuracy by combining information from the rest of the sentences. Figure 4 shows an illustration of this combination process. LeanContext keeps the top- $k$  sentences intact, while other sentences between the top- $k$  sentences are reduced by an open-source text reduction method. LeanContext maintains the order of the top- $k$  sentences and other sentences according to their appearances in the original context in order to produce more accurate results.

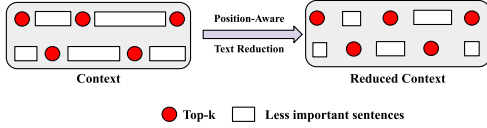


Figure 4: Illustration of the reduction of less important sentences while keeping the top- $k$  sentences intact in a context.

The idea of maintaining the top sentences intact is a simple but interesting approach. Adding this simple approach to the existing open-source pre-trained summarizer models will help increase the performance. However, an interesting question lies in identifying the  $k$  in top- $k$ . We ask the question here, (a) *What is the number of  $k$ ?* We answer the question in a more detailed manner with a reinforcement learning (RL) based solution as follows.

---

Algorithm 1: LeanContext Training Algorithm

---

**Require:**  $\mathcal{D}$ : Input documents

**Require:**  $q$ : A set of queries

**Require:**  $\Theta$ : a set of predefined thresholds, acts as a set of actions in RL

**Ensure:**  $Q^*$ : Trained  $Q$ -table

```

1:  $\mathcal{X} \leftarrow \emptyset$  ▷ set of states
2: for each  $q, \text{context} \in q, \mathcal{D}$  do
3:    $\mathbf{v}_c, \mathbf{v}_q \leftarrow \text{Embedding}(\text{context}), \text{Embedding}(q)$  ▷ get embedding vectors
4:    $\mathcal{X} \leftarrow \mathcal{X} \cup (\mathbf{v}_c - \mathbf{v}_q)$ 
5: end for
6:  $\mathcal{S} \leftarrow K\text{-Means}(\mathcal{X})$  ▷ centroids as state vectors
7: for each  $q, \text{context} \in q, \mathcal{D}$  do
8:    $\text{state} \leftarrow \text{get\_state}(\mathcal{S}, \text{context}, q)$ 
9:    $\text{action} \leftarrow \text{get\_action}(\Theta)$ 
10:   $\mathbf{C} \leftarrow \text{retrieve}(q, \mathbf{v}_q, \theta)$  ▷ context formation
11:   $\mathbf{C}' \leftarrow \text{perform\_action}(\mathbf{C}, \text{action})$  ▷ text reduction
12:   $\text{answer} \leftarrow \text{llm}(q, \mathbf{C}')$ 
13:   $r \leftarrow \text{compute\_score}(\text{answer}, \text{original\_answer})$ 
14:   $\text{reward} \leftarrow \alpha(2r - r^*) - (1 - \alpha) \times \tau(\mathbf{C}', \mathbf{C})$ 
15:   $Q(\text{state}, \text{action}) \leftarrow Q(\text{state}, \text{action}) + \frac{1}{n}(\text{reward} - Q(\text{state}, \text{action}))$  ▷ Update  $Q$  table
16: end for
17: return  $Q^*$  ▷ return trained  $Q$  table

```

---

**Adaptive- $k$ :** Identifying the number  $k$  is crucial for context reduction as well as performance. Considering a fixed- $k$  might impact accuracy if  $k \ll n$ . On the contrary, if  $k \approx n$ , the token ratio will be higher, leading to higher costs. hence, to achieve minimal cost with maximum accuracy, the  $k$  should be adaptive. To get a query-based adaptive context, we propose a lightweight  $Q$ -learning-based reinforcement learning algorithm that perceives an optimal policy for an agent operating in an environment. After training, we will have an optimal  $Q$  table that takes the best action for a given state.

$$\pi^*(\text{state}) = \underset{a}{\operatorname{argmax}} Q^*(\text{state}, \text{action})$$

**Details of RL:** Due to the dynamic environment of domains and queries, it is difficult to estimate the optimal context. In this situation where the environment is complex and dynamic, RL fits the best. After retrieving the context based on a query, LeanContext computes the state with context and query embedding. With the state, the RL agent finds a suitable action from the trained  $Q^*$  table. Based on the action, the threshold for context reduction is computed and top- $k$  sentences are selected. Then the reduced context version is produced according to one of the sentence positioning variants. After then the response is retrieved from the reduced context and query and sent to the LLM. We carefully define our state, action, and reward function as follows.

**State:** We combine query and context to define the *state*. At offline profiling, we compute the embedding of the query  $\mathbf{v}_q$  and the embedding of the context as  $\mathbf{v}_c$ . Then we subtract  $\mathbf{v}_q$  from  $\mathbf{v}_c$  that indicates the context-query pair. After building the vector with a number of training samples, we run the  $K$ -means model to compute the centroids. These centroids are utilized as state vectors  $\mathcal{S}$ . So, at run time, a query context pair is concatenated using their embedding vectors and the closest centroid will be the state for them.

$$\mathcal{S} \leftarrow K\text{-means} \left( \bigcup_{i,j} (\mathbf{v}_{c_i} - \mathbf{v}_{q_j}) \right)$$

**Action:** As our goal is to make the top- $k$  extraction adaptive. Given a set of thresholds from 0 to 0.4 (maximum top- $k$  will be 40% of the total context) to select the number of sentences, we define each possible choice as an action of the proposed RL system. The outcome of each action choice is computed by the reward function and the  $Q$  table is updated for the corresponding (state, action) pair.

**Reward:** The reward for the RL model is higher if the context ratio is less and the accuracy is almost equal to the optimal accuracy using full context. We compute the ROUGE score ( $r$ ) to evaluate the answer using reduced context with the actual answer using full context. If the ROUGE score with expensive LLM is ( $r^*$ ) for a query, then the current (state, action) will be rewarded if  $r - r^* \geq 0$ , otherwise will be penalized. For the token ratio, the lower the better as the reduction of context as much as possible without compromising accuracy is rewarded. Thus, the reward function  $\mathbf{R}$  is defined as follows.

$$\mathbf{R} = -(1 - \alpha)\tau + \alpha(2r - r^*)$$



**Training algorithm:** The off-policy  $Q$  table training algorithm is shown in Algorithm 1. In line 1 – 6, each state is computed by the subtraction of query embedding from context embedding. A k-means model is trained to get the centroids and utilize the centroids as different states. In line 7 – 15 for selecting each threshold as action, the corresponding reward is computed the  $Q$  table is updated. Finally, the updated  $Q$  table is deployed for reducing the context. The training of RL requires LLM to compute the reward. To reduce the training cost, we perform training on fewer samples. hence to observe the effect of each action in each state, we do a full exploration to update the  $Q$  table.

**LeanContext Inference** The LeanContext inference algorithm is shown in Algorithm 2. For each query, the corresponding context is retrieved, and the state is computed by the trained RL-agent [line 2]. Based on the state, the threshold  $\theta$  is computed as an action to select the top-k sentences and the top-k sentences with reduced less important sentences produce the reduced context,  $C'$  [line 3-4]. This reduced context is utilized to get answer with the LLM [line 5]. Finally, the answer is returned to the user.

---

Algorithm 2: LeanContext Inference

---

**Require:**  $\mathcal{D}_t$ : Test documents

**Require:**  $q_t$ : A set of test queries

**Require:** *Agent*: Trained RL-Agent

```

1: for each  $q, context \in q_t, \mathcal{D}_t$  do
2:    $state \leftarrow Agent.get\_state(Agent.S, context, q)$ 
3:    $action \leftarrow Agent.get\_action(state)$ 
4:    $C' \leftarrow Agent.perform\_action(context, action)$ 
5:    $answer \leftarrow llm(q, C')$ 
6:   return  $answer$ 
7: end for

```

---

## Experimental Settings

**Dataset:** In real-time scenarios, user documents are new to LLMs such as gpt-3.5-turbo model, and thus should not be able to answer a query without giving a context. To ensure this, we use recent arxiv papers and BBC news articles so that the LLMs are not trained on these. Following this, we use the [Arxiv Dataset and BBC News Dataset](#) where documents are published in March 2023 (Li 2023). We generate questions with answers for each document using QAGenerationChain from LangChain (Harrison Chase 2022) based on gpt-3.5-turbo model.

**Baseline Models:** In our question-answering-based system, we mainly focus on reducing the context length while keeping the same performance in QA. So, we use a [pay-per-use LLM \(gpt-turbo-3.5\) model](#) for all cases to answer a query based on the context. We evaluate our proposed context length reduction approach with the recent context reduction approaches as follows.

1. Context (Original): We keep the context length intact and ask LLM to answer the query.
2. CQSumDP (Laskar et al. 2023): We generate the following prompt similar to CQSumDP to generate the query-

aware summary of a context.

“A document along with its query is given below. Write down the most reasonable summary relevant to its document-query pair.

Document: {CONTEXT}

Query: {QUERY}”

3. Semantic Compression (Gilbert et al. 2023): A query unaware prompt is stated to compress a context as follows. “Please compress the following text into a latent representation that a different gpt-3.5-turbo model can decompress into the original text. The compression model should purely minimize the number of characters in the compressed representation while maintaining the semantics of the original text. The resulting compressed text does not need to be decompressed into the original text but should capture the semantics of the original text. The compressed text should be able to be decompressed into a text that is semantically similar to the original text but does not need to be identical. Text to Compress: {CONTEXT}”
4. SBert (Miller 2019): A Bert-model-based extractive summarization approach. The context is reduced to several sentences. We keep the number of sentences 3 in our experiments.
5. Selective Context (SC (Li 2023)): It utilizes entropy to filter out less informative content from context. In our experiments, we employ phrase-level content filtering with different reduction ratios. We use GPT-2 model to compute the self-information.
6. Flan-T5-Base (Chung et al. 2022): We use this 250M encoder-decoder model to summarize the context based on the same instruction as CQSumDP.

**Implementation Details:** To implement the document ingestion, we use a cheap all-MiniLM-L6-v2 model (Reimers and Gurevych 2019) as an embedding generator. The text chunks along with the embeddings are stored in ChromaDB (ChromaDB) vector database. We use the chunk size 500, chunk overlap is 0. We vary the number of chunks  $N = 2, 4, 8, 10$  to compare how well the RL algorithm performs. We use this template for the QA: “*Answer to the question based on the given context. Context: {CONTEXT}, Question: {QUERY}, if you do not find any answer in the context, simply return ‘No answer’*”. We use [LLMChain from Langchain \(Harrison Chase 2022\)](#) to ask queries to the [OpenAI model by prompting](#). LeanContext uses selective context method (Li 2023) to reduce less important sentences by 80%.

## Results

**Arxiv Dataset:** We consider random 25 articles from the Arxiv dataset for testing and another 5 documents RL training distinct from the 25 documents. After retrieving the query, we use the RL agent to identify to top-k in the context and reduce the context using the RL agent. For the same query, and context setting, we evaluate our approach with the baseline approaches. The comparison of LeanContext with baseline models is shown in Table 1. LeanContext achieves

Text Reduction Method	Avg. Total tokens	Avg. Prompt tokens	Avg. Summary tokens	Avg. Completion tokens	ROUGE-1	ROUGE-2	ROUGE-L	Cost Savings (%)
Context (Original)	547	521	0	26	<b>0.3985</b>	<b>0.2868</b>	<b>0.3714</b>	0.00
CQSumDP	631	89	517	25	<b>0.4424</b>	<b>0.3061</b>	<b>0.4213</b>	<b>-15.36</b>
Semantic Compression	939	319	597	23	0.3331	0.2221	0.3132	-71.66
T5-base	79	71	0	8	0.1614	0.1101	0.1486	85.56
SBert	205	188	0	17	0.2563	0.1701	0.2469	62.52
SC (reduction = 0.50)	334	316	0	18	0.2945	0.2014	0.2755	38.94
LeanContext (Fixed k = 0.1)	210	196	0	14	0.2305	0.1623	0.2173	61.62
LeanContext (Adaptive k [RL])	343	321	0	22	<b>0.3844</b>	<b>0.2684</b>	<b>0.3577</b>	<b>37.29</b>

Table 1: Comparison on Random 100 samples from Arxiv Dataset. Number of chunks,  $N = 4$

Text Reduction Method	Avg. Total tokens	Avg. Prompt tokens	Avg. Summary tokens	Avg. Completion tokens	ROUGE-1	ROUGE-2	ROUGE-L	Cost Savings (%)
Context (Original)	761	724	0	37	<b>0.5498</b>	<b>0.4172</b>	<b>0.5337</b>	0.00
CQSumDP	842	75	738	29	<b>0.5801</b>	<b>0.4405</b>	<b>0.5637</b>	<b>-10.64</b>
Semantic Compression	1078	228	820	30	0.4729	0.3204	0.4517	-41.66
T5-base	74	51	0	23	0.3993	0.2631	0.3752	90.28
SBert	174	147	0	27	0.4261	0.2917	0.4082	77.14
SC (reduction = 0.50)	461	429	0	32	0.4740	0.3308	0.4521	39.42
LeanContext (Fixed k = 0.1)	278	250	0	28	<b>0.5017</b>	<b>0.3740</b>	<b>0.4872</b>	<b>63.47</b>
LeanContext (Adaptive k [RL])	245	218	0	27	<b>0.5233</b>	<b>0.3943</b>	<b>0.5093</b>	<b>67.81</b>

Table 2: Comparison on Random 100 samples from BBC-News Dataset. Number of chunks,  $N = 8$

similar performance compared to the original context with no text reduction and outperforms existing open-source models with 37% cost savings. CQSumDP (Laskar et al. 2023) achieves better accuracy with a greater cost (adds 15.36% more cost) by reducing the minimal context to get the right answer than the original context. Throughout the experiments, we observe the same effect and conclude that the zero-shot performance of GPT-4 like LLMs is better with concise and relevant context to query (CQSumDP) than the context with a lot of irrelevant information. Although Semantic Compression (Gilbert et al. 2023) uses LLM to minimize context, due to the context-agnostic nature of the prompt, it performs even worse than LeanContext and adds additional  $\sim 72\%$  cost.

**BBCNews Dataset:** We consider random 100 news articles from the BBCNews dataset. We keep 80 articles for testing and the rest 20 articles to train the RL agent. For query-based context retrieval settings, we evaluate our approach with the baseline approaches.

Due to the cost of the OpenAI model and the current usage limit, we follow recent literature (Yang et al. 2023), we generate queries using the QA generation method, and consider random 100 query samples for evaluation. The evaluation result is shown in Table 2.

We observe that generating a query-aware summary from document-query pair using OpenAI LLM (Laskar et al. 2023) performs better i.e. ROUGE-1 0.5801 than query-aware open-source baseline methods such as T5-base i.e. 0.3993 but with a high cost. However, it also contributes more cost than the original context (10.64% more). We also observe that the query-unaware LLM using a different hard prompt (Gilbert et al. 2023) even performs worse i.e. ROUGE-1 score 0.4729 with a 41.66% cost overhead. Additionally, adding top- $k(0.1)$  uplifts the ROUGE-1 score of T5-base by  $\sim 12.35\%$ . With adaptive top- $k$  (Top- $k$ (RL)) and adaptive top- $k$  (Top- $k$ (RL)) with sentence order, it performs even better by reducing the cost from 65%  $\sim 74\%$ . We observe the same scenario for other models too. Further investigation with different number of chunks for the same

Dataset	Text Reduction Method	Avg. Total tokens	Avg. Prompt tokens	Avg. Summary tokens	Avg. Completion tokens	ROUGE-1	ROUGE-2	ROUGE-L	Cost Savings (%)
Arxiv	Context (Original)	547	521	0	26	<b>0.3985</b>	<b>0.2868</b>	<b>0.3714</b>	<b>0.00</b>
	T5	79	71	0	8	0.1614	0.1101	0.1486	85.56
	T5 + LeanContext (Only Top-k=0.1)	131	113	0	18	0.3325	0.2390	0.3146	76.05
	T5 + LeanContext (Only Top-k=RL)	284	263	0	21	0.3942	0.2847	0.3742	48.08
	T5 + LeanContext	357	335	0	22	<b>0.4073</b>	<b>0.2863</b>	<b>0.3809</b>	<b>34.73</b>
	SBert	205	188	0	17	0.2563	0.1701	0.2469	62.52
	SBert + LeanContext (Only Top-k=0.1)	250	230	0	20	0.3104	0.2181	0.2949	54.30
	SBert + LeanContext (Only Top-k=RL)	405	380	0	25	0.3676	0.2594	0.3464	25.96
	SBert + LeanContext	478	452	0	26	<b>0.3885</b>	<b>0.2720</b>	<b>0.3596</b>	<b>12.61</b>
	LeanContext	343	321	0	22	<b>0.3844</b>	<b>0.2684</b>	<b>0.3577</b>	<b>37.29</b>
BBCNews	Context (Original)	761	724	0	37	<b>0.5498</b>	<b>0.4172</b>	<b>0.5337</b>	0.00
	T5	74	51	0	23	0.3993	0.2631	0.3752	90.28
	T5 + LeanContext (Only Top-k=0.1)	142	116	0	26	0.5228	0.3914	0.5065	81.34
	T5 + LeanContext (Only Top-k=RL)	192	165	0	27	0.5368	0.4072	0.5187	74.77
	T5 + LeanContext	259	232	0	27	<b>0.5532</b>	<b>0.4290</b>	<b>0.5374</b>	<b>65.97</b>
	SBert	174	147	0	27	0.4261	0.2917	0.4082	77.14
	SBert + LeanContext (Only Top-k=0.1)	241	212	0	29	0.5072	0.3731	0.4914	68.33
	SBert + LeanContext (Only Top-k=RL)	289	260	0	29	0.5200	0.3827	0.5039	62.02
	SBert + LeanContext	355	327	0	28	<b>0.5355</b>	<b>0.4007</b>	<b>0.5235</b>	<b>53.36</b>
	LeanContext	245	218	0	27	<b>0.5233</b>	<b>0.3943</b>	<b>0.5093</b>	<b>67.81</b>

Table 3: Effect of cascading LeanContext with open source summarizers.

test queries are discussed in the Ablation Study section.

**Top-k is all you need:** We observe an interesting phenomenon when adding our LeanContext with existing open-source models. As these open-source models are not trained on new domain data, we observe that combining 10% top- $k$  sentences with the open-source models boosts the QA system’s performance by 5.41%  $\sim 17.11\%$  for the Arxiv dataset and 8.11%  $\sim 12.35\%$  for the BBCNews dataset as shown in Table 3. In addition, using top- $k$  on the fly with a generic summarizer adds an extra advantage to the overall system. Our LeanContext with top- $k$  sentences using RL and reduced version of other sentences in between them make the performance even better than the fixed (10%) top- $k$  by 13.22%  $\sim 24.59\%$  for the Arxiv dataset and 10.94%  $\sim 15.39\%$  for the BBCNews dataset (Table 3). LeanContext with T5-base model outperforms all the existing approaches including no reduction method.

Text Reduction Method	Avg. Total tokens	Avg. Prompt tokens	Avg. Summary tokens	Avg. Completion tokens	ROUGE-1	ROUGE-2	ROUGE-L	Cost Savings (%)
<b>Number of chunks, N=2</b>								
Context (Original)	243	211	0	32	0.5370	0.3987	0.5190	0.00
CQSumDP	322	70	225	27	0.5303	0.3897	0.5105	-32.51
Semantic Compression	448	113	307	28	0.4786	0.3169	0.4519	-84.36
T5-base	76	50	0	26	0.4010	0.2640	0.3781	68.72
SBert	157	128	0	29	0.4825	0.3395	0.4604	35.39
SC (reduction = 0.50)	164	137	0	27	0.4614	0.3113	0.4403	32.51
LeanContext	117	92	0	25	<b>0.4556</b>	<b>0.3265</b>	<b>0.4373</b>	<b>51.85</b>
<b>Number of chunks, N=4</b>								
Context (Original)	417	384	0	33	<b>0.5489</b>	<b>0.4183</b>	<b>0.5327</b>	0.00
CQSumDP	496	71	398	27	0.5569	0.4156	0.5389	-18.94
Semantic Compression	665	159	479	27	0.4916	0.3328	0.4684	-59.47
T5-base	74	50	0	24	0.3921	0.2555	0.3678	82.25
SBert	163	136	0	27	0.4626	0.3287	0.4452	60.91
SC (reduction = 0.50)	263	235	0	28	0.4726	0.3250	0.4507	36.93
LeanContext	153	128	0	25	<b>0.4856</b>	<b>0.3605</b>	<b>0.4700</b>	<b>63.31</b>
<b>Number of chunks, N=10</b>								
Context (Original)	930	892	0	38	0.5423	0.4072	0.5246	0.00
CQSumDP	1008	74	906	28	0.5549	0.4440	0.5644	-8.39
Semantic Compression	1274	258	987	29	0.4862	0.3343	0.4642	-36.99
T5-base	74	51	0	23	0.3903	0.2584	0.3688	92.04
SBert	181	153	0	28	0.4164	0.2807	0.3967	80.54
SC (reduction = 0.50)	559	527	0	32	0.4825	0.3312	0.4591	39.89
LeanContext	279	252	0	27	<b>0.5318</b>	<b>0.4069</b>	<b>0.5205</b>	<b>70.00</b>

Table 4: Evaluation of RL model on a different number of chunks on a random 100 samples from BBCNews Dataset. Our RL agent trained with ( $N=8$ ) shows promising results while applying on different chunk numbers.

## Ablation Study

We investigate whether the RL model has a performance dependency on the number of chunks. So, we train the RL model using  $N = 8$  and run the same model on  $N = 2, 4$ , and 10. The results are shown in Table 4. We observe that although the  $N$  changes, the RL model still outperforms other less expensive baseline methods. Another interesting obser-

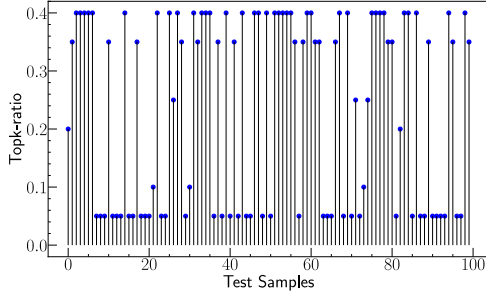


Figure 5: Adaptive- $k$  ratio selected by the BBCNews RL agent based on queries.

Compression Method	Avg. Total tokens	Avg. Prompt tokens	Avg. Completion tokens	ROUGE-1	ROUGE-2	ROUGE-L
None	538	514	24	0.3945	0.2904	0.3764
LeanContext (state = $\frac{\mathbf{v}_c \cdot \mathbf{v}_q}{\ \mathbf{v}_c\  \ \mathbf{v}_q\ }$ )	296	275	21	0.3443	0.2369	0.3259
LeanContext (state = $\mathbf{v}_c - \mathbf{v}_q$ )	331	308	23	0.3553	0.2535	0.3388
LeanContext (state = $\mathbf{v}_c \oplus \mathbf{v}_q$ )	284	265	19	0.3400	0.2370	0.3223

Table 5: Comparison of RL state functions on Arxiv dataset

vation lies in making  $N$  as larger as possible for the successful retrieval of context using LeanContext whereas previously without LeanContext  $N$  should be kept smaller to reduce the cost of LLM usage. In Figure 5, we show how the action (top- $k$  ratio) is taken by the RL agent given the context and query for each query sample out of 100 samples. Considering an adaptive top- $k$  ratio chosen by our RL agent varies over query samples to achieve the adaptive reduction of context.

We also empirically evaluate state function with different associations between context ( $\mathbf{v}_c$ ) and query embedding ( $\mathbf{v}_q$ ) i.e. cosine similarity (state =  $\frac{\mathbf{v}_c \cdot \mathbf{v}_q}{\|\mathbf{v}_c\| \|\mathbf{v}_q\|}$ ), concatenation ( $\mathbf{v}_c \oplus \mathbf{v}_q$ ), and subtraction ( $\mathbf{v}_c - \mathbf{v}_q$ ). Among them, subtraction performs best (Table 5). In this experiment, we only select the top- $k$  sentences using RL to evaluate the impact of state definition on performance.

## Conclusion and Future Work

In this paper, we propose LeanContext, a cost-efficient query-aware context reduction system to reduce the cost associated with LLM API usage. Despite the reduction of prompt tokens, LeanContext achieves similar or better performance compared to the no reduction of the original context. The advantage of top- $k$  is that it can be plugged in with any existing summarization method of a domain-based QA system to boost the overall performance according to our experimental results. Here, we only focus on text-based context as a domain. We will explore other domains in our future work.

## References

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Chen, L.; Zaharia, M.; and Zou, J. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *arXiv preprint arXiv:2305.05176*.

ChromaDB. 2023. ChromaDB. <https://www.trychroma.com/>. Accessed: June 20, 2023.

Chung, H. W.; Hou, L.; Longpre, S.; Zoph, B.; Tay, Y.; Fedus, W.; Li, E.; Wang, X.; Dehghani, M.; Brahma, S.; et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Gilbert, H.; Sandborn, M.; Schmidt, D. C.; Spencer-Smith, J.; and White, J. 2023. Semantic Compression With Large Language Models. *arXiv preprint arXiv:2304.12512*.

GPT-3 Cost. 2023. GPT-3 cost estimation for real applications. <https://neoteric.eu/blog/how-much-does-it-cost-to-use-gpt-models-gpt-3-pricing-explained/>. Accessed: July 11, 2023.

Harrison Chase. 2022. LangChain. <https://github.com/hwchase17/langchain>. 2022-10-17.

Jiao, W.; Wang, W.; Huang, J.-t.; Wang, X.; and Tu, Z. 2023. Is ChatGPT a good translator? A preliminary study. *arXiv preprint arXiv:2301.08745*.

Laskar, M. T. R.; Rahman, M.; Jahan, I.; Hoque, E.; and Huang, J. 2023. CQSumDP: A ChatGPT-Annotated Resource for Query-Focused Abstractive Summarization Based on Debatepedia. *arXiv preprint arXiv:2305.06147*.

Li, Y. 2023. Unlocking Context Constraints of LLMs: Enhancing Context Efficiency of LLMs with Self-Information-Based Content Filtering. *arXiv preprint arXiv:2304.12102*.

Ling, C.; Zhao, X.; Lu, J.; Deng, C.; Zheng, C.; Wang, J.; Chowdhury, T.; Li, Y.; Cui, H.; Zhao, T.; et al. 2023. Beyond One-Model-Fits-All: A Survey of Domain Specialization for Large Language Models. *arXiv preprint arXiv:2305.18703*.

Liu, Y.; Han, T.; Ma, S.; Zhang, J.; Yang, Y.; Tian, J.; He, H.; Li, A.; He, M.; Liu, Z.; et al. 2023. Summary of chatgpt/gpt-4 research and perspective towards the future of large language models. *arXiv preprint arXiv:2304.01852*.

Luo, Z.; Xie, Q.; and Ananiadou, S. 2023. Chatgpt as a factual inconsistency evaluator for abstractive text summarization. *arXiv preprint arXiv:2303.15621*.

Ma, C.; Wu, Z.; Wang, J.; Xu, S.; Wei, Y.; Liu, Z.; Guo, L.; Cai, X.; Zhang, S.; Zhang, T.; et al. 2023. ImpressionGPT: an iterative optimizing framework for radiology report summarization with chatGPT. *arXiv preprint arXiv:2304.08448*.

Miller, D. 2019. Leveraging BERT for extractive text summarization on lectures. *arXiv preprint arXiv:1906.04165*.

Pinecone. 2023. Vector database. <https://www.pinecone.io/learn/vector-database/>. Accessed: June 20, 2023.

Reimers, N.; and Gurevych, I. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Schlag, I.; Sukhbaatar, S.; Celikyilmaz, A.; Yih, W.-t.; Weston, J.; Schmidhuber, J.; and Li, X. 2023. Large language model programs. *arXiv preprint arXiv:2305.05364*.

Tan, Y.; Min, D.; Li, Y.; Li, W.; Hu, N.; Chen, Y.; and Qi, G. 2023. Evaluation of ChatGPT as a question answering system for answering complex questions. *arXiv preprint arXiv:2303.07992*.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837.

Wu, J.; Antonova, R.; Kan, A.; Lepert, M.; Zeng, A.; Song, S.; Bohg, J.; Rusinkiewicz, S.; and Funkhouser, T. 2023. Tidybot: Personalized robot assistance with large language models. *arXiv preprint arXiv:2305.05658*.

Yang, X.; Li, Y.; Zhang, X.; Chen, H.; and Cheng, W. 2023. Exploring the limits of chatgpt for query or aspect-based text summarization. *arXiv preprint arXiv:2302.08081*.

Zhang, H.; Liu, X.; and Zhang, J. 2023a. Extractive summarization via chatgpt for faithful summary generation. *arXiv preprint arXiv:2304.04193*.

Zhang, H.; Liu, X.; and Zhang, J. 2023b. SummIt: Iterative Text Summarization via ChatGPT. *arXiv preprint arXiv:2305.14835*.