Efficient Long Text Summarization Using an sLLM Pipeline

Byeongjin Kang* **HoJae Kim*** **HunTae Kim*** **Joonyeol Choi*** **Minsu Kim*** **Saehun Chun***
Sungkyunkwan University
{qudwlskbj, ghghghost, huntae324, joonyeol99, kms48491000, saehun0519} @ g.skku.edu

ABSTRACT

This paper explores the development of a compact, edge-deployable Large Language Model (LLM) for summarizing video lecture transcripts. Motivated by the need for efficient learning from online video content, we initially pursued a direct fine-tuning approach using a small LLM (1B-7B parameters). However, resource constraints related to maximum token length and dataset distribution discrepancies led to suboptimal performance. Consequently, we propose a novel segmentation-based approach, dividing transcripts into smaller, semantically related segments using cosine similarity derived from sentence embeddings. These segments are then clustered into themes using various methods, including timeline-based concatenation, KNN, and DBSCAN, before being individually summarized by a specialized summarization LLM with 500 million parameters. This segmented approach, coupled with the use of a significantly smaller, specialized LLM, allows for efficient summarization of long video transcripts, addressing the limitations of traditional LLMs on resource-constrained devices. Evaluation using ROUGE and BERTScore demonstrates the effectiveness of our proposed method compared to baseline approaches and highlights the potential for efficient video lecture summarization on edge devices. Code and results can be found on the project repository https://github.com/SKKUWhiteBoard/WhiteBoard_LLM.

1 Introduction

The proliferation of online video lectures presents a valuable opportunity for accessible education. However, efficiently extracting key information from these often lengthy videos remains a challenge. This necessitates the development of automated summarization tools that can condense video content into concise and informative summaries, facilitating efficient learning and review. Our initial approach focused on fine-tuning a small Large Language Model (LLM) for this task. This approach, motivated by the desire for an edge-deployable solution suitable for personal computers and other resource-constrained devices, leveraged readily available open-source LLMs (1B-7B parameters).

Our first attempt involved directly fine-tuning the chosen LLM using a dataset of YouTube lecture transcripts paired with summaries generated by GPT-4o-mini. This approach, while conceptually straightforward, encountered significant challenges. The average length of the YouTube transcripts, measured in tokens, far exceeded the maximum input length manageable by the LLM during fine-tuning, given our available computational resources on Kaggle T4 GPUs. This discrepancy between training and test data distributions, coupled with the inherent limitations of smaller LLMs on long sequences, resulted in suboptimal summary quality, including issues such as LLM collapse and inadequate semantic preservation.

This initial setback prompted a shift in our methodology. We pivoted to a segmentation-based approach, hypothesizing that breaking down long transcripts into smaller, semantically coherent units would mitigate the limitations encountered previously. Crucially, this revised approach incorporates a much smaller, specialized summarization LLM with 500 million parameters, a substantial reduction compared to the billion-parameter models used initially. The subsequent sections of this paper detail our proposed segmentation method, the various clustering algorithms explored for grouping related segments, and the final summarization process using this specialized, compact LLM.

2 Method

2.1 Segmentation & Embedding

First, the task of this part is dividing the whole long text into segments. This is controlled by two parameters: "n_word" which defines how many words each segment should consist of, and "n_overlap" which determines how many words should overlap between adjacent segments. Additionally, the "fix_size" parameter is used to handle cases where the last segment is shorter than expected.

The reasons for separating the segments using the number of words, not the number of sentences or letters, are as follows. If cut based on the number of letters, meaningless tokens could be generated depending on the tokenizer characteristics. And, if we use the number of sentences, dividing by sentence count could lead to significant variability in the length of each segment. Therefore, the decision was made to use word count to ensure that each segment is of a similar length while preserving the contextual meaning that the LLM can process effectively.

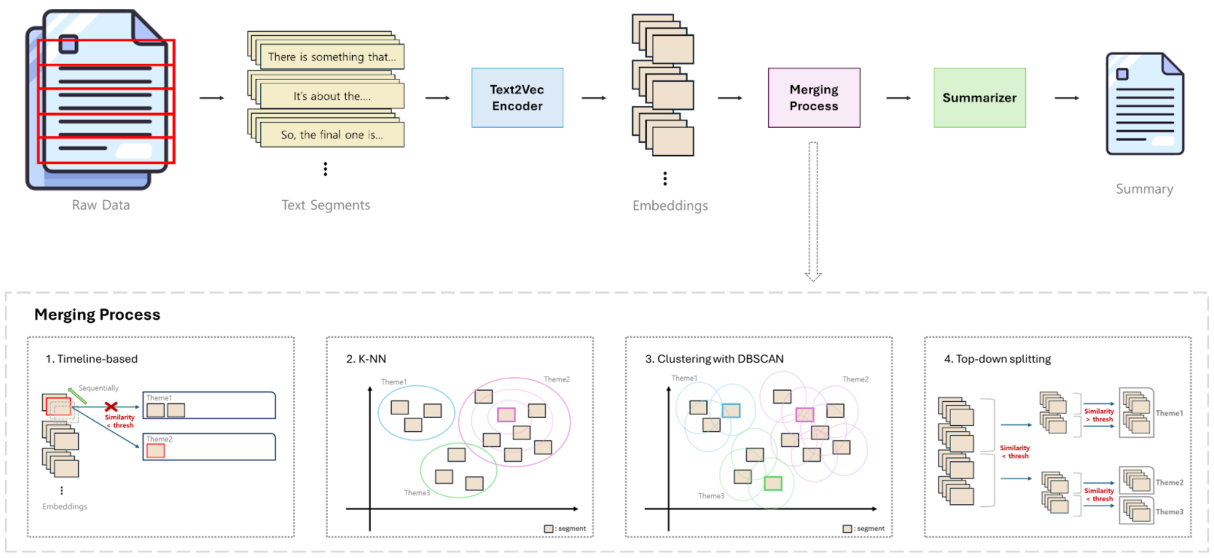

Figure 1: Figure Caption and Image above the caption [In draft mode, Image will not appear on the screen]

Here, each segment is embedded as one vector. This is to compute each similarity in the subsequent merging step to bundle similar content into the theme buffer. In this project, we use the "all-MiniLM-L6-v2" model provided by "sentence-transformer". This model has a size of 22.7M and can perform embedding in a very short amount of time, which fits well with the objectives of the project. However, a user can do the embedding using any model freely.

2.2 Merging

The process of calculating the similarity between each segment is performed using the embedding vector of the segments obtained in 2.1. Any method that can calculate the similarity between vectors, such as cosine similarity, dot product, and euclidean distance, can be used for calculation. Cosine similarity was used in this project.

Now, based on the calculated similarity, segments with similar meanings are regrouped into one theme. There can be many methods of this grouping process, but here we will introduce some of them we implemented and experimented with.

Timeline-based

The concat_timeline_based method groups segments into themes by sequentially comparing the similarity between consecutive segments. Segment embeddings are generated using the encode_segments function. The similarity between the i-th and (i+1)-th embeddings is calculated using a similarity metric like cosine similarity.

If the similarity exceeds a predefined threshold, the segments are grouped into the same theme. Otherwise, a new theme group is created. The output is a list of groups, each representing a thematic cluster of consecutive segments. This method preserves the order of segments and ensures thematically similar segments are grouped together.

K-NN

The concat_knn method uses the k-Nearest Neighbors (k-NN) algorithm to cluster embeddings based on pairwise similarity. A K-NearestNeighbors model is initialized with k, the number of neighbors to consider, and the embeddings are processed to identify their nearest neighbors. Neighbors are iteratively examined for each embedding, and those within the similarity threshold are grouped together. A tracking mechanism ensures that embeddings already assigned to a group are not reassigned. Groups are formed based on similarity, each groups containing indices of closely related embeddings. This method outputs a list of groups, where each group represents a cluster of embeddings that are highly similar based on the distance threshold. Unlike DBSCAN, this approach does not explicitly handle noise and assumes all embeddings are part of some group or form standalone clusters when no neighbors meet the threshold.

Clustering (DBSCAN)

The concat_clustering method utilizes the DBSCAN algorithm to group segments based on their embeddings. DBSCAN is a density-based clustering approach that identifies clusters as dense regions in the embedding space while classifying sparse points as noise. DBSCAN is then applied with two primary parameters: eps, which defines the maximum distance between two points to be considered part of the same cluster, and min_samples, which specifies the minimum number of points required to form a dense region. Each embedding is assigned a cluster label, with noise points labeled as -1. Groups of embeddings are then formed based on their cluster labels, with each cluster corresponding to a list of indices. Noise points are collected separately. The output is a list of groups, containing the indices of embeddings belonging to the same cluster.

Clustering (Hierarchical)

The concat hierarchical clustering method uses a hierarchical clustering algorithm to group text segments based on their embeddings. A linkage matrix is then calculated using a selected linkage method (e.g., 'ward') with a distance metric (e.g., 'euclidean'). Flat clusters are formed by cutting the hierarchical tree at a specified distance threshold, dynamically grouping embeddings based on their proximity in the embedding space.

Unlike methods like KNN, hierarchical clustering does not require the number of clusters (k) to be predefined, making it suitable for documents with varying thematic structures. Each cluster corresponds to a list of indices representing the grouped embeddings, and the output is a list of these groups. This approach is particularly useful for diverse datasets as it adapts to the distribution of the data without the need for a fixed cluster count.

Top-down splitting

Above merging approaches is the process of grouping starting from the smallest segment level. On the other hand, this top-down splitting approach works by dividing from the whole text level.

First, starting with the entire text, the target text is split into two parts, embeddings are generated for these splitted parts, and their similarities are compared. If the similarity is smaller than a certain threshold, the two split segments are assumed to represent different content and are treated as a new target text. The above process is repeated until the whole text segments are merged with a similarity greater than the threshold or reaches the minimum size of segments.

2.3 Summarizing

After similar contents are grouped into a theme buffer, each group is then summarized using a summarizer sLLM model, and these summaries are integrated to create an overall summary of the entire text. The summarizer model, like the embedding model, can be any model, but for the project's goal of using a light-weight small LLM to summarize texts that exceed the context length limit, the "bart-large-cnn" model was used. This model has 406M parameters and was trained on the CNN DailyMail dataset for summarization tasks.

Our code is implemented to call the summarizer in a batch inference. This allows for the fast generation of an overall summary, even when composed of a large number of themes. During experiments, it was observed that when the number of themes was very large, the inference speed significantly decreased due to exceeding hardware resource (VRAM) limits. As a result, the batch for all themes was split into mini-batches for inference, further confirming the feasibility of usage in a local environment.

3 Experiments: Metric Introduction

In this section, we introduce the evaluation metrics used to assess the performance of the proposed summarization approach. These metrics are widely used in natural language processing (NLP) tasks to quantify the quality of generated summaries by comparing them with reference texts. We employ the following metrics:

3.1 ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE is a standard metric for evaluating text summarization quality. It measures the overlap between the n-grams of the generated summary and the reference summary. The key variants used are:

- **ROUGE-1**: Measures the overlap of individual words (unigrams).

$$ROUGE\_1 = \frac{Number\ of\ overlapping\ unigrams}{Total\ unigrams\ in\ reference\ summary}$$

- **ROUGE-2**: Measures the overlap of bigrams (two consecutive words).

$$ROUGE\_2 = \frac{Number\ of\ overlapping\ bigrams}{Total\ bigrams\ in\ reference\ summary}$$

- **ROUGE-L**: Uses the Longest Common Subsequence (LCS) to capture both order and content overlap.

$$ROUGE\_L = \frac{LCS\ length}{Reference\ summary\ length}$$

These metrics provide a comprehensive evaluation of both content coverage (recall) and conciseness (precision).

3.2 BERTScore

BERTScore evaluates the semantic similarity between the generated and reference summaries by leveraging contextual embeddings from a pre-trained BERT model. It calculates the cosine similarity between token embeddings, capturing nuanced semantic relationships.

$$\text{BERTScore} = \frac{1}{|\text{T}|}\sum_{\text{t}\in\text{T}} \max_{\text{r}\in\text{R}} \text{cosine\_sim}(\text{Embedding}(\text{t}), \text{Embedding}(\text{r}))$$

Where:

- T represents tokens in the generated summary.
- R represents tokens in the reference summary.

Key advantages:

- Captures semantic meaning beyond surface-level word matching.
- Handles paraphrasing effectively.

Table 1: Evaluation Results

Dataset	Avg. length	Concat Method	ROUGE(1/2/L)	BERTScore
YouTube	30k	Timeline-based	**62.68/59.21/57.91**	**75.72 (±3.99)**
		knn	9.41/8.80/8.91	65.60 (±8.89)
		Clustering	**61.32/57.83/55.61**	**72.92 (±6.78)**
Gov-report	60k	Timeline-based	33.85/31.64/30.44	**73.31 (±6.76)**
		Clustering	**75.60/69.71/27.61**	72.92 (±6.78)

Table 2: Experiment Results via n-word

Evaluation Metric	50-word segment	150-word segment	300-word segment
ROUGE-1	76.057	62.679	29.676
ROUGE-2	71.239	59.214	27.435
ROUGE-L	68.698	57.915	27.357
BERTScore	77.479	73.671	66.859

3.3 Semantic Similarity

Semantic similarity evaluates the meaning of the generated summary compared to the reference summary. It uses sentence-level embeddings (e.g., from models like Sentence-BERT or Sent2Vec) and computes the cosine similarity between the embeddings of the two summaries:

$$\text{Semantic}_{\text{Similarity}} = \text{sim}_{\text{cosine}(\text{Embedding}_{\text{Generated}},\text{Embedding}_{\text{Reference}})}$$

This metric provides a deeper understanding of whether the generated summary conveys the same overall meaning as the reference, even if the wording or structure differs significantly.

4 Evaluation Result

In this experiment, we aimed to address the challenge of segmenting and concatenating meaningful text segments from long texts to facilitate processing by small language models (sLLM) with limited input length capacity. The datasets used in this study include the YouTube dataset, with an average segment length of 30k characters, and the Government Report dataset, which has an average segment length of 60k characters. For each dataset, 100 random samples were selected, and the results (f1-score) were scaled by a factor of 100 for ease of interpretation.

Dataset

The YouTube dataset showed varying performance depending on the concatenation method used. Among the methods tested, the best results achieved a ROUGE score averaging around 60, indicating strong performance in summarizing thematic content. Additionally, the BERTScore for this dataset reached up to 75, showcasing the ability to capture semantic similarity effectively.

For the Government Report dataset, which consists of longer and more complex text, similar trends were observed. While the performance varied across different algorithms, the results were consistent in achieving competitive scores. The BERTScore reached values around 73, which is notable considering the challenges posed by such lengthy texts.

Comparative Analysis

When compared to large-scale models like GPT, which typically achieve BERTScores in the range of 85 to 95, this approach demonstrates that small language models can still achieve approximately 70% of the performance. This is significant, considering the constraints on computational resources.

Analysis of ROUGE Scores

The high ROUGE scores can be attributed to the n-gram-based calculation method. During the concatenation process, if the final merged themes lead to a higher number of thematic segments, the summaries generated may include repetitive information. This repetition across different themes contributes to inflated ROUGE scores as the metrics favor overlap in content.

The content in Table 2 presents the ROUGE and BERTScore performance based on the number of words (n-word) constituting each segment in the timeline-based concatenation method. It is observed that smaller values of n-word lead to better performance across all evaluation metrics. This trend can be attributed to the fact that when the number of words in each segment is reduced, the size of the theme also tends to be smaller, resulting in clearer semantic representation.

Additionally, experiments were conducted by varying the similarity threshold in the timeline-based method. It was observed that higher threshold values resulted in better performance across all evaluation metrics. This outcome suggests that, during the clustering process of segments forming each theme, only those with high similarity are grouped together. As a result, both the structural score, measured by ROUGE, and the semantic score, measured by BERTScore, show improved performance when higher similarity thresholds are applied.

In addition to the results presented in the table, various experiments were conducted on different merging methods with respect to several parameters like similarity threshold, eps (DBSCAN), k (K-NN), etc.

Clustering (DBSCAN): The variation in scores based on the epsilon value in the DBSCAN method showed minimal significance. In contrast, experiments conducted with respect to the minimum cluster size, min_samples, revealed that smaller values resulted in higher performance. This can be interpreted as follows: when the number of segments within each theme is smaller, it becomes easier to maintain both the structural and semantic integrity between the original text and the summary, leading to improved results.

Top-down splitting: Experiments were conducted in the Top-down splitting method with respect to both similarity threshold and n-word. First, in the experiment with varying threshold values, a significant performance improvement in both ROUGE and BERTScore was observed when the threshold was set above 0.75. This can be attributed to the fact that, similar to the timeline-based method, segments with similar semantics tend to group together. In the experiment with n-word, it was observed that as the value of n-word increased, the scores for all metrics slightly decreased. This suggests that as the size of the segments grows, it becomes more challenging to maintain the structure and semantics of the original text in the summary.

K-NN: Experiments were conducted by varying the value of k in the K-NN method. In these experiments, BERTScore showed minimal variation with changes in k, while a slight increase in ROUGE Score was observed as k decreased.

5 Conclusion

This paper introduces a novel segmentation-based framework for summarizing lengthy video lecture transcripts using a small Large Language Model (sLLM) with 500 million parameters. By segmenting transcripts into semantically coherent units and clustering related segments through methods like timeline-based concatenation, KNN, and DBSCAN, the proposed approach addresses key challenges such as token length limitations and computational constraints. This strategy enables efficient summarization on resource-constrained edge devices, achieving performance comparable to much larger LLMs like GPT models without requiring fine-tuning.

The primary contributions of this work are threefold. First, it demonstrates an innovative and scalable summarization framework that significantly reduces computational requirements while maintaining high performance, making it suitable for deployment on edge devices. Second, it introduces a flexible methodology that allows users to control the granularity of summaries by adjusting segmentation and clustering parameters, accommodating diverse application needs. Third, it establishes a cost-effective and practical solution for real-world use cases, including video content summarization, document compression, and real-time indexing, paving the way for broader adoption of sLLMs in similar tasks.

6 Discussion and Limitations

While the approach has some limitations, such as the evaluation reliance on metrics like ROUGE and BERTScore, which may not fully capture semantic nuances, and the lack of external knowledge incorporation due to the smaller model size, these are outweighed by the framework's strengths. Future enhancements, such as integrating retrieval-augmented generation (RAG) architectures or replacing embedder and summarizer models, could address these minor limitations. Overall, this study demonstrates that small, specialized LLMs, when paired with intelligent pre-processing techniques, can achieve remarkable performance, offering a transformative approach to efficient and scalable summarization in resource-constrained settings.

REFERENCES

[1] Huang, L., Cao, S., Parulian, N., Ji, H., & Wang, L. (2021). Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*.

[2] Cao, S., & Wang, L. (2022). HIBRIDS: Attention with hierarchical biases for structure-aware long document summarization. *arXiv preprint arXiv:2203.10741*.

[3] Lin, C. Y. (2004, July). Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out* (pp. 74-81).

[4] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2019). Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.