Tree-Based Retrieval Using Gaussian Statistics

Stanford CS224N {Custom} Project

Luke Park
Department of Mathematics
Stanford University
luke2004@stanford.edu

Irfan Nafi
Department of Physics
Stanford University
inafi@stanford.edu

Ray Hotate
Department of Computer Science
Stanford University
rayhtt@stanford.edu

Abstract

Retrieving long-form text for question-answering LLMs is a difficult problem to address since maintaining computational efficiency while still capturing context is a tough balance to strike. In this paper, we address this problem by expanding upon RAPTOR's novel tree method (Parth Sarthi, 2024). After implementing a baseline model of the RAPTOR model, we found two large limitations. Firstly, through experimentation, we found that the naive Gaussian model mixture model used in the paper leads to cluster boundaries that are ill-defined. Furthermore, at inference time the tree traversal relies solely on cosine similarity which is not a good metric especially considering that we embed chunks of text (Faisal et al., 2012). To address this we augmented the features being passed into the clustering algorithm and created a heuristic for traversal which overcame the described limitations. Moreover, to create less bias in our results, we intentionally used the weaker Mistral-7B model to show that the context our method provides was truly relevant. The results were promising, despite using a weaker model, on the QuALITY benchmark we were within 10% accuracy of the RAPTOR model which is the current state-of-the-art model and uses GPT-4 at inference. We also beat out every other non-GPT-based model on the QuALITY leaderboard.

1 Key Information to include

- Mentor: Tony Wang
- External Collaborators (if you have any): None
- Sharing project: Not sharing
- Ray: Clustering Algorithms, Embeddings, Analysis, Luke: TF-IDF, Evaluation Metrics, Experiments, Irfan: Tree Building, Traversal

2 Introduction

Consider the question "How does Percy Jackson embody being a hero?" Large language models (LLMs) such as ChatGPT have been trained on the Percy Jackson books and due to the sheer size of the model, it comes up with a good answer citing character traits and some of the trials that Percy faces. However, what happens to pieces of text that are published after the model's training? It is computationally infeasible to run the training process again. For relatively short texts the solution is to feed the entirety of the text with the query so that the LLM has context to answer the question. Unfortunately, this is not a generalizable solution since an incoming text corpus could be larger than the context window. Even though context windows are rapidly growing, so does the compute required, as attention is quadratic in complexity.

With these considerations in mind, our goal is to limit the amount of context we provide for answering a paper, efficiently retrieve from our external database the k-best short chunks of text, and still represent longer forms of information. This problem has two primary technical challenges: condensing information with minimal loss and creating a structure that allows for

efficient retrieval. Such a solution has massive implications for problems such as academic literature search, which is a field where the corpus is always increasing, giving rise to queries that are technically dense and abstract, requiring a level beyond that of keyword-based search.

We base our work on the recursive summarizing idea from the RAPTOR paper (Parth Sarthi, 2024), which clusters chunks of text, creates a summary on each cluster, re-clusters on embedding created from the new summaries, and generates more summaries where each summary is a parent to the constituents of its cluster building up a tree from the bottom up.

Our contribution comes from the realization that the clustering algorithm and the traversal used in the RAPTOR paper were lacking. The Gaussian model used just the sentence embeddings and we improved upon it by adding a relative positional encoding which allowed the clustering model to qualitatively nudge sentences in proximity closer to each other while allowing the embeddings to also semantically cluster. Furthermore, we added an additional parameter in the traversal, creating a TF-IDF score that allows for traditional keyword-based search to have some sway over the purely semantic-based search coming from the cosine similarity. This will help with esoteric or infrequent terminology that may be particularly important to a query.

3 Related Work

RAGs have seen improvements in the retriever starting off with traditional term-based techniques like **TF-IDF** (Sang-Woon and Joon-Min, 2019) moving on into deep learning-based strategies. For more end-to-end approaches encoder-decoder models have been combined with bidirectional masked LLM's fine-tuned for question answering (Gao et al., 2024).

Surprisingly though, the majority of work for RAGs has not been focused on solely the retrieval part. This is due to the fact that rapid innovations in the LLM size were allowing for poorer forms of retrieval to still show progress. But, with the rapid innovation in stand-alone lightweight models such as Mistral-7B being able to perform very well on numerous benchmarks, we are of the belief that the now more important factor in the improvement of RAG's performance will come from retrieval.

4 Approach

4.1 Overview

In order to efficiently search for granular matches, the text first needs to be broken down into smaller sequences. These sequences are the leaf nodes, which are embedded, and then clustered together, followed by a summary generated on these clusters using a LLM. The summary acts as the parent to the sequences in its clusters, at which points the parents are also clustered, repeating this process as many times as needed until no more parents are needed. After the summarization, we add a one-hot encoding of the keywords using TF-IDF so that at traversal keyword search can also be used as part of a heuristic function. In Figure 3, we illustrate this process in detail.

4.2 Baseline

The baseline model is the implementation directly from the RAPTOR paper using a naive clustering algorithm with no additional information beyond the embeddings being passed, and tree traversal solely using cosine similarity.

4.3 Term Frequency - Inverse Document Frequency (TF-IDF)

For TF-IDF, we created a binary vector (whose length is the number of keywords we have), where each index refers to a specific keyword. The value at an index is 1 if that keyword is in the query or sentence and 0 if it isn't. Another thing to note is that the binary vector is only for words that pass a threshold score. This is because we are only interested in the occurrence of rare words. We came up with this threshold by observing the TF-IDF formula:

$$\text{TF-IDF} = \frac{\text{Number of times } t \text{ appears in } d}{\text{Total number of terms in } d} \times \log \frac{\text{Number of documents}}{1 + \text{Number of documents word appears in}} \tag{1}$$

After testing on rare words, we found a threshold of 0.3 to be an optimal value for keywords that represent relative rarity. The motivation behind using TF-IDF is that while traditional keyword search is limited; if a particular region of the tree contains a keyword and has semantic similarity, it should be weighed more strongly. Also, we want to create better summaries by using TF-IDF to identify the keywords in a cluster and prompt Mistral-7B to use the keywords in its summaries which could potentially increase the cosine similarity during traversal making a branch more promising.

4.4 Clustering

To aggregate embeddings with similar meanings (so that summaries have as little loss of information as possible), we clustered the embeddings after reducing the dimensions. We experimented with multiple clustering methods, namely K-Means and Gaussian Mixture Models, as shown below.

4.4.1 K-Means

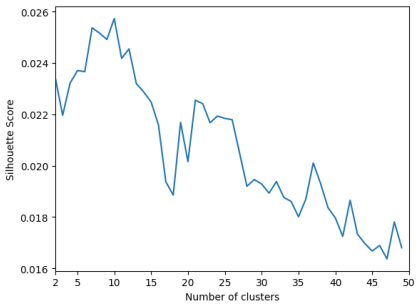


Figure 1: Clustering optimization for an article in the data, showing the silhouette score per cluster number.

For our first clustering approach, we used a standard K-Means algorithm to cluster the embedding with the reduced dimensionality from Principal Component Analysis (PCA). We then searched for the optimal number of clusters to use, by using the silhouette score to inform us about clustering quality. The silhouette score serves as a measure of how similar an element is to its own cluster compared to other clusters. The silhouette value for a single sample is a ratio that can range from -1 to 1, where a high value indicates that the object is well-matched to its own cluster and poorly matched to neighboring clusters.

However, K-Means has some shortcomings, specifically with its assumption of normally distributed data that form nice, spherical clusters (Huang et al., 2014). Furthermore, because embeddings are not perfect representations of the semantic meanings of sentences, assigning sentences to just one cluster may not be ideal when traversing the tree, as some sentences may get assigned to the wrong cluster and will thus never be explored.

4.4.2 Gaussian Mixture Models (GMM)

GMMs are better at modeling clusters with different shapes and sizes (He et al., 2011). We incorporated GMMs by following the steps below:

1. **Dimensionality Reduction with Uniform Manifold Approximation and Projection (UMAP)**
   We used UMAP to reduce the dimension of embedding (Leland McInnes, 2020). The embedding had the 2d-array shape of (1339, 384), and we reduced it to the shape of (1339, 2).
2. **Determining the Optimal Number of Clusters using Bayesian Information Criterion (BIC)**
   The GMM model was optimized by testing various combinations of clusters, from 1 to 100, and covariance types, such as spherical, tied, diagonal, and full. For each pair, the Expectation-Maximization (EM) algorithm was adopted to have it fit a GMM, calculate

the BIC, which penalizes the model complexity while rewarding according to how well the model fits (Schwarz, 1978), and then update the smallest BIC values.

For the number of clusters n and the number of sentences m, the distribution of GMM is as follows:

$$P(x) = \sum_{k=1}^{n} \pi_k N(x|\mu_k, \Sigma_k) \tag{2}$$

$$N(x|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{\frac{m}{2}} |\Sigma_k|^{\frac{1}{2}}} \exp(-\frac{1}{2}(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k)) \tag{3}$$

x : The reduced embeddings

μ_k : The $1 \times m$ mean vector for the k-th gaussian distribution

Σ_k : The $m \times m$ covariance matrix for the k-th gaussian distribution

π_k : The weight for the k-th gaussian distribution

We calculated μ_k, Σ_k, and π_k using the EM algorithm.
The BIC was calculated with the following formula:

$$\text{BIC} = -2 \log L + M \log n \tag{4}$$

L : The maximum of likelihood function

M : The number of parameters that define the model, including μ_k's, Σ_k's, and π_k's

The number of clusters n such that the BIC score achieves its minimum is the optimal n.

3. **Clustering with the best GMM model**
With the optimal GMM model, clustering was executed. The output was the list of the probability distributions for each text segment indicating which cluster it should belong to. Table 1 shows the clustering that is based on the highest probability of belonging for each sentence. The left figure shows that the data points, representing sentences, that are close to one another share the same cluster to belong to with the highest probability. The right figure additionally shows the ellipsoids of all the Gaussian distributions.

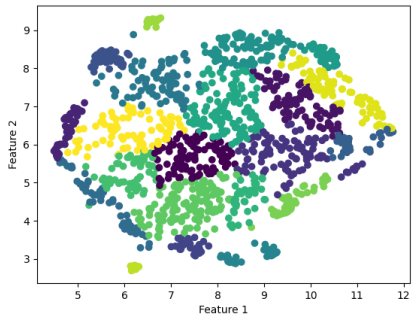
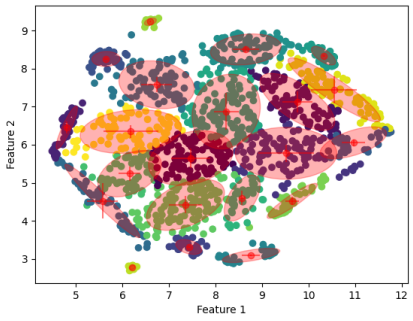
Without Ellipsoids	With Ellipsoids
	

Table 1: Clustering Assignments for the Highest Probability

4. **Sentences to Cluster assignment**
Each sentence was distributed into clusters with the following procedure. First, place each of the sentences in a cluster that it is most likely to belong to. Next, iterate again through every sentence, and for each sentence, if the membership probability for some other cluster was over the 10% threshold and that cluster has a size less than 75%, place it in that cluster. Thus, text segments may be assigned to multiple clusters. This allows the result of the clustering to grasp the nature that a single sentence can be relevant to different topics, represented by different clusters while resolving the disparity of the sizes of different clusters.

4.5 Building the Tree

The tree starts with the leaf nodes, each of which contains an extracted sentence, embedding, binary TF-IDF vector, and index within the data. For the first level, the embeddings for the leaf nodes are clustered, as mentioned above, such that each cluster contains a summary of the sentences within the cluster. To generate the summary, we used the Mistral-7B v0.2 model (Jiang et al., 2023). If there are too many sentences in a cluster, the prompt for summarizing the sentences will exceed the input limit of Mistral (8k tokens for this model), so we randomly sample 80 sentences from the cluster, as each sentence is around 100 tokens. With this summary, we create a new node representing this cluster. For the new node's embedding vector, we run the embedding model on the summary and for the TF-IDF vector, we run a union of all of the TF-IDF vectors of the nodes in the cluster.

This process is repeated until there are less than 25 nodes in a level, at which point a root node is created (that summarizes the remaining nodes) to make traversal easier. Finally, to traverse the tree, we find the most relevant leaf nodes by recursively maximizing the cosine similarity and TF-IDF score (as shown in 9) between the query embedding and the node embedding until the leaf node is found.

4.6 Traversal

For traversal, we first generate embeddings for the query. Then, for each level of the tree, we find the node with the highest score for our traversal function, which incorporates the cosine similarity of the embeddings and the Jaccard similarity of the TF-IDF binary vectors (for binary vectors we take the bitwise-and for union and bitwise-or for intersection, for which we then square to get the sum of the vectors). For two nodes, u, and v, with embeddings vectors, x_u and x_v, respectively, and TF-IDF binary vectors, t_u and t_v, respectively, the score is calculated as follows:

$$\textsc{Score}(u, v) = \alpha \frac{\vec{x_u} \cdot \vec{x_v}}{||\vec{x_u}||\,||\vec{x_v}||} + \beta \left(\frac{|\vec{t_u} \wedge \vec{t_v}|}{|\vec{t_u} \vee \vec{t_v}|} \right)^2 \tag{5}$$

(α, β are hyperparameters to be tuned later.) The traversal ends when a leaf node is reached, however, because the nodes in the path to the leaf node offer a higher-level understanding of the text through summarization, we also return these summaries with the leaf node.

4.7 Positional Information

As for the final step, we add the positional information to the embeddings and tune the hyperparameters. Positional information ensures that the sequence order is preserved, which is essential for understanding context and meaning in sentences.
Let $\mathbf{w}_i$ represent the vector embedding of the i-th sentence ($i = 0, 1, \ldots m$, where m is the number of sentences). The embedding with positional information, $\mathbf{E}_{new}$, is presented as the following formula:

$$\mathbf{E}_{new} = \begin{bmatrix} \mathbf{P}^\top \\ \mathbf{E} \end{bmatrix} \tag{6}$$

$$\mathbf{P} = [\mathbf{p_E} \quad \mathbf{p_E} \quad \cdots \quad \mathbf{p_E}] \tag{7}$$

$$\mathbf{p_E} = \begin{bmatrix} \mu_\mathbf{E} - k\sigma_\mathbf{E} \\ \mu_\mathbf{E} - k(1 - \frac{2}{m})\sigma_\mathbf{E} \\ \mu_\mathbf{E} - k(1 - \frac{4}{m})\sigma_\mathbf{E} \\ \vdots \\ \mu_\mathbf{E} + k(1 - \frac{2}{m})\sigma_\mathbf{E} \\ \mu_\mathbf{E} + k\sigma_\mathbf{E} \end{bmatrix} \tag{8}$$

$\mathbf{E}$: The original embedding matrix with $\mathbf{w}_i$ as its i-th column

$\mathbf{P}$: $m \times l$ matrix with vector $\mathbf{p_E}$ for every column, where l is the number of $\mathbf{p_E}$'s added to E

$\mu_\mathbf{E}$: The mean of all the elements in the matrix $\mathbf{E}$

$\sigma_\mathbf{E}$: The standard deviation of all the elements in the matrix $\mathbf{E}$

k : The coefficient of $\sigma_\mathbf{E}$ that determines the range of the position values in $\mathbf{p_E}$

If $l = 1$, the positional information for a single sentence is the same weight as a single element in the embedding vector of that sentence, and it will be negligible. The hyperparameter l is made to allow the positional information to increase its weight in the matrix. Another hyperparameter k, the coefficient for $\sigma_{\mathbf{E}}$, represents to what extent it differentiates the position of sentences in the entire text. It makes sure that the mean of the elements in $\mathbf{p_E}$ is same as the mean of all the elements in $\mathbf{E}$, while adopting $\sigma_{\mathbf{E}}$ for the normalization process.
The hyperparameter tuning was conducted by varying k, l and running the steps above.

5 Experiments

5.1 Data

We used the QuALITY dataset (Richard Yuanzhe et al., 2022). It is comprised of 6,737 multiple-choice questions on passages with an average length of 5,000 tokens and an average of 10 questions per passage (there are 700 passages total). We are not doing any training in the traditional sense, but we were hyperparameter tuning along with swapping out different components of the model. As such we constructed a training set in which we randomly sampled 1500 questions across a total of 140 passages and tested how many questions each model got correctly. For final evaluation the test dataset has 155 passages, and a total of 1585 test questions.

5.2 Evaluation method

Unlike generation, retrieval unfortunately does not have a direct method of measuring its effectiveness. Thus, we used a standard proxy measure of using an LLM to take the results from the retrieval model we created and answer the questions from the test set of the QuALITY dataset. Initially, we were going to test our model by taking the given text for each prompt and creating the tree. However, this was too simple a task since each text is only 5,000 tokens, and even the Mistral-7B model we were using has enough context to answer the problem with the entire text passed in. Thus to make the task more challenging and more generalizable, we created a tree with all 155 passages for the test, and all 140 passages for the training set. This is a more difficult task since retrieval now has to be done in a tree where the wanted passages are mixed in with passages that do not pertain to the question. Note that the test set accuracy is a measurement of percent: 84.5 is 84.5%. Additionally, we also recorded the average answer time per question since a naive approach that would work but is inefficient is to simply search through all chunks.

5.3 Experimental details

Our experiment pipeline was straightforward. For each evaluation, we swapped out the clustering algorithm and traversal algorithm as needed and ran the tree-building model. We then used the evaluation method above to see how the model performed on the test set. For comparisons of the clustering algorithm, we isolated a traversal method so that it was clear that the clustering algorithm was the driving force of the results. For the traversal method, we did the same and isolated the clustering algorithm

5.4 Results

First, without positional information emphasis, K-Means and GMMs were compared. As a control the naive traversal method was used, and Table 2 was obtained.

Clustering Algorithm	Number of Clusters	Training Set Accuracy	Avg. Time per Question
K-Means	10	33.3	39.4 secs
K-Means	20	39.4	26.5 secs
K-Means	30	38.7	24.3 secs
GMMs	10	53.3	38.94 secs
GMMs	20	60.1	27.7 secs
GMMs	30	58.7	24.3 secs

Table 2: Clustering model comparison

Since the GMMs turned out to outperform, the embedding with the positional information added was tested with GMMs. The tuning for hyperparameters (k, l) used in adding positional information to the embedding was conducted, whose full result is shown in Table 5. The optimal pair (k, l) was $(1.3, \frac{L}{2})$, with 25 as the number of clusters and spherical covariance type.

Having determined the best clustering algorithm to use, we now performed hyperparameter tuning to figure out the ideal values of α, β from the traversal score formula:

$$\text{SCORE}(u, v) = \alpha \frac{\vec{x_u} \cdot \vec{x_v}}{||\vec{x_u}|| \, ||\vec{x_v}||} + \beta \left(\frac{|\vec{t_u} \wedge \vec{t_v}|}{|\vec{t_u} \vee \vec{t_v}|} \right)^2 \tag{9}$$

Using the GMM model of 25 clusters with $k = 1.3$, $l = \frac{L}{2}$, and a spherical covariance type (these are the ideal parameters which we will discuss more in the analysis), we got:3

Traversal Algorithm	α	β	Training Set Accuracy	Avg. Time per Question
Naive	NA	NA	62.4	29.4 secs
Heuristic	0.95	0.05	65.3	26.5 secs
Heuristic	0.80	0.20	69.3	24.3 secs
Heuristic	0.75	0.25	72.5	28.94 secs
Heuristic	0.72	0.28	74.1	28.94 secs
Heuristic	0.70	0.30	70.1	27.7 secs
Heuristic	0.60	0.40	56.4	24.3 secs

Table 3: Traversal Algorithm Comparison

5.5 Comparative Results

Our baseline model can be thought of as the direct implementation of what the RAPTOR paper laid out. But to also give a more comparative result of the impressive nature of what we accomplished in our final model we laid out the top 4 performers of the QuALITY dataset leaderboard and where we would stand 4.

Model Name	LLM used	Testing Set Accuracy
Raptor	GPT-4	82.6
Baseline model	Long-context GPT-3.5	74.7
Document Retrieval with Gaussian Statistics	**Mistral-7B**	**73.9**
LongMA	TechGPT-7B	73.0
Naive Raptor	Mistral-7B	62.4

Table 4: QuALITY Leaderboard

6 Analysis

6.1 GMM Clustering

The appendix A.2 includes Table 6 showing the visualized result of tuning the hyperparameters (k, l), which is how the distribution changes through the change of those two values. Figure 2 shows the distribution with the optimal result. The optimal result was achieved when the embedding had a weak encoding of the positional information. Refer to Appendix A.2 for further comparison and the relationship between the numerical results and the distribution.

6.2 Heuristic Function Hyperparameter Search

A key insight we made that significantly decreased the search time for the ideal values was adding the following constraint: $\alpha + \beta = 1$. This is mathematically sound because scores are

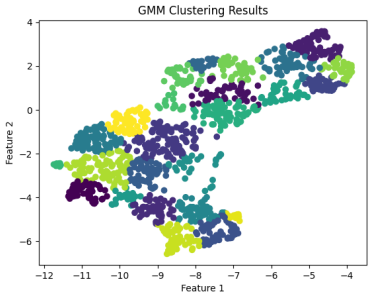


Figure 2: Clustering distribution for the optimal result

being compared against each other, and by capping $\alpha + \beta = 1$, there isn't an infinite search space for the values of α, β. We also said that $\alpha, \beta \geq 0$ since heuristically cosine similarity and TF-IDF should increase the score.

6.3 Where We Fall Short

The QuALITY dataset has additional tags on questions showing what type of reasoning the question requires. By far our weakest performing groups were the symbolism/interpretation reasoning type, and the finish the phrase type of reasoning. Our most likely explanation for the reduced performance in these two categories is that symbolism/interpretation requires a higher level of reasoning that is simply lacking in a model such as Mistral-7B. The limiting factor here is likely the LLM and not the retrieval method. On the other hand, finishing the phrase type of questions can be improved upon by fine-tuning the LLM beforehand. Also, expanding the number of outputs could increase the probability of finding a segment of text that could help answer these types of questions since this type of question is more about threading the needle than long-form retrieval.

7 Conclusion

In this paper, we expanded upon the novel RAPTOR tree method, identifying and addressing its limitations through enhanced clustering algorithms and heuristic traversal. Our approach introduced significant improvements in the Gaussian model mixture and cosine similarity metrics, augmented by positional information and TF-IDF scoring.

Our experimental results, utilizing the weaker Mistral-7B model for testing, demonstrated promising accuracy on the QuALITY benchmark, showcasing our method's efficacy even with less powerful LLMs. This achievement not only validates the relevance of the context provided by our method but also positions our work competitively among state-of-the-art models, including those based on the more capable GPT-4.

There are still many challenges to overcome, as this system is very reliant on accurate embeddings and has difficulties handling questions with complex or nuanced reasoning. However, because of our system's efficiency, we believe it is worth further developing this work.

7.1 Future Work

Our project is highly modular, with the major areas being: pre-processing, clustering, summarization, tree building, and traversal. As we refactor our model to address the task of retrieval on research papers, we want to improve each of the core areas. For pre-processing, there is not much more to do. But for clustering, we want to incorporate additional metadata such as a rolling summary to give further contextualization for a given sequence. Also for the tree building, if the corpus reaches a certain size, we need to do hyperparameter tuning to figure out what the optimal number of layers and branching factors is which also affects how the clustering is done. Finally, for the traversal, we wish to add a Monte-Carlo Tree Search (Świechowski et al., 2023) to roll out the most promising nodes of the tree at a given level and to better evaluate its future output. For our project though with the size of our corpus, since our tree was only 4 layers, we thought MC Tree Search was overkill.

References

Rahutomo Faisal, Kitasuka Teruaki, and Aritsugi Masayoshi. 2012. Semantic cosine similarity. In *The 7th international student conference on advanced science and technology ICAST. Vol. 4. No. 1.*, pages 1–2.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey.

Xiaofei He, Deng Cai, Yuanlong Shao, H. Bao, and Jiawei Han. 2011. Laplacian regularized gaussian mixture model for data clustering. *IEEE Transactions on Knowledge and Data Engineering*, 23:1406–1418.

Xiaohui Huang, Yunming Ye, and Haijun Zhang. 2014. Extensions of kmeans-type algorithms: A new clustering framework by integrating intracluster compactness and intercluster separation. *IEEE Transactions on Neural Networks and Learning Systems*, 25:1433–1446.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

James Melville Leland McInnes, John Healy. 2020. Umap: Uniform manifold approximation and projection for dimension reduction. Online. arxiv.org.

Aditi Tuli Shubh Khanna Anna Goldie Christopher D. Manning Parth Sarthi, Salman Abdullah. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. Online. ICLR.

Pang Richard Yuanzhe, Parrish Alicia, Joshi Nitish, Nangia Nikita, Jason. Phang, Chen Angelica, Padmakumar Vishakh, Ma Johnny, Thompson Jana, He He, and Bowman Samuel. 2022. Quality: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, page 5336–5358, Online. Association for Computational Linguistics.

Kim Sang-Woon and Gil Joon-Min. 2019. Research paper classification systems based on tf-idf and lda schemes. Online. Human-centric Computing and Information Sciences.

Gideon Schwarz. 1978. Estimating the dimension of a model. Online. Project Euclid.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2023. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497.

A Appendix

A.1 Architecture

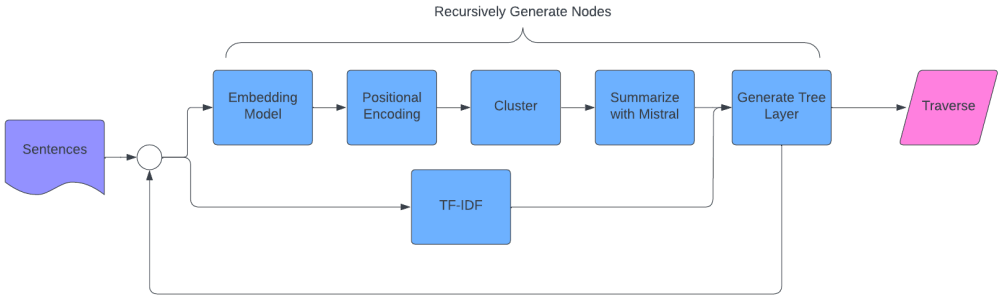


Figure 3: An architecture diagram of our system, showing how data is processed to recursively generate the tree.

A.2 Tuning Hyperparameters for Positional Information and Determining the Best GMMs

As Table 6 shows, the distribution of the clustering (as was done in the GMM process) was plotted for different k's and l's. To compare the ratio between l and L, representing the length of the original embedding vector for each sentence, different values of l were tested as the function of L. As the values of k, l increase, the position data is emphasized. This is because the increase of k means that the positional difference of two sentences gets larger, and the increase of l means that the weight of the positional information compared to the original embedding vectors gets larger. An intensive increase of k, l decreased the score since there was too much emphasis on the positional information. This is seen in Table 6, where there was a strong emergence of a curve with large k, l. The plot with $(k, l) = (1.3, \frac{L}{2})$, the optimal result, showed the weak emergence of a curve diverging from the original uniform distribution. For the hyperparameter tuning, we used a GMM with 25 clusters, and the naive traversal in order to control and show that the change in performance was strictly coming from the selection of k and l.

		Coefficients of standard deviation (k)		
		0.7	1.3	2
Length of adding position information (l)	$L/4$	55.5	57.3	58.8
	$L/2$	58.7	**62.4**	60.8
	L	56.9	61.4	57.4

Table 5: Tuning on the k, l

Table 7 presents the BIC scores correlated with varying cluster counts from 1 to 100 for each covariance pattern on the condition of $(k, l) = (1.3, \frac{L}{2})$. The number of clusters is represented on the x-axis, and the BIC scores on the y-axis.

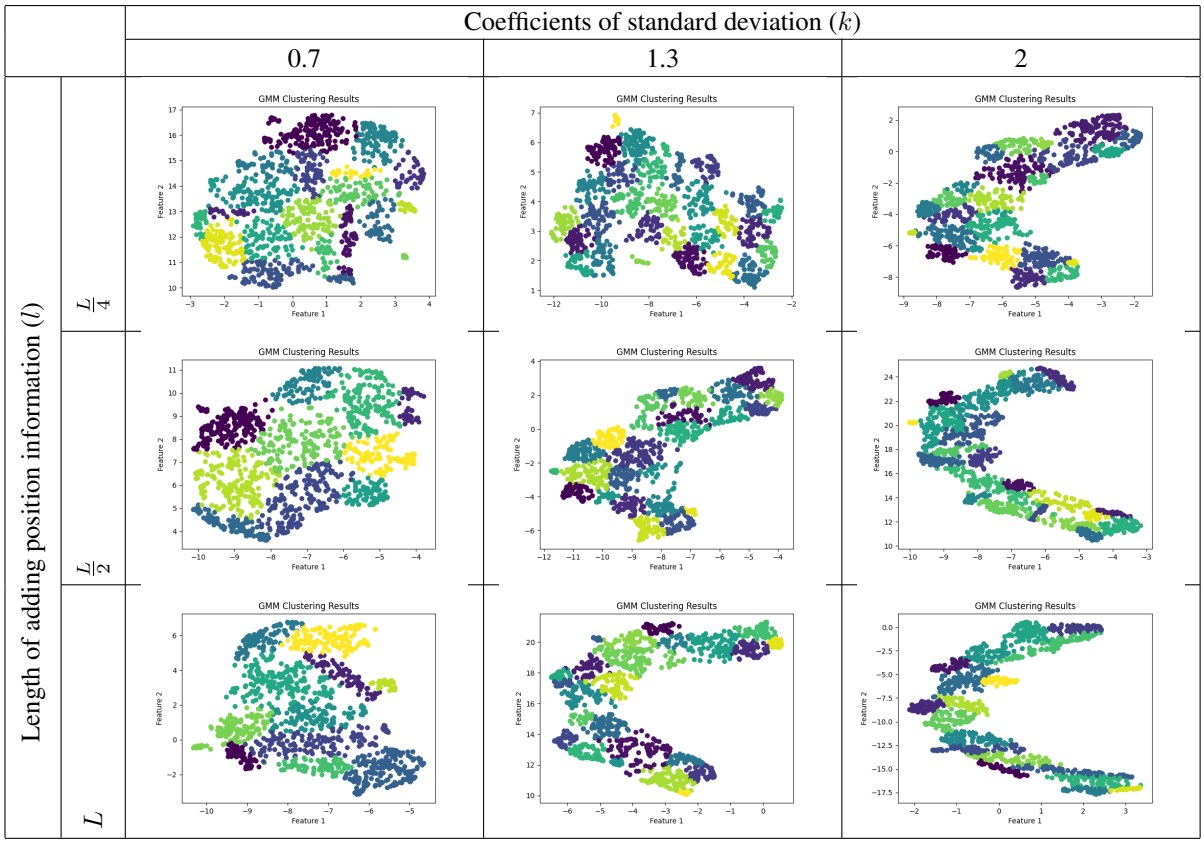


Table 6: Clustering Distributions with Positional Information

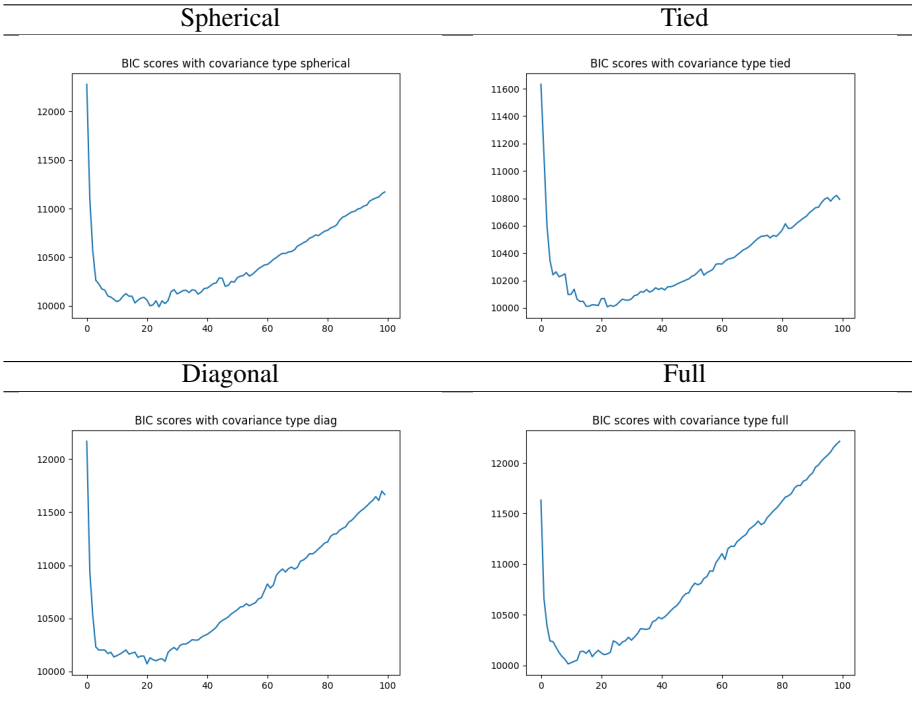


Table 7: Covariance Type Comparison for BIC scores